NASA Technical Memorandum 102224

The 3DGRAPE Book: Theory, Users' Manual, Examples

Reese L. Sorenson

(NASA-TM-102224) THE 3DGRAPE BOOK: THEORY, N89-25116
USERS' MANUAL, EXAMPLES (NASA. Ames
Research Center) 122 p CSCL 01B
Unclas
G3/01 0219856

July 1989

NASA
National Aeronautics and
Space Administration

NASA Technical Memorandum 102224

The 3DGRAPE Book: Theory, Users' Manual, Examples

Reese L. Sorenson, Ames Research Center, Moffett Field, California

July 1989

NASA
National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

CONTENTS

Page

Abstract 1

I. Introduction 1

II. The program: its availability and installation 4

  A. Tape file 1: the 3DGRAPE program 4

  B. Tape file 2: input data for example cases 9

III. Zoning 10

  A. A primer on zoning 10

  B. Zoning with 3DGRAPE 14

IV. Input 19

  A. The first two lines 19

  B. File10 – control scalars for new start 19

    1. The "run-comment" lines 20

    2. The "number-of-blocks" line 21

    3. The "iterations" lines 22

    4. The "filename-11" line 23

    5. The "filename-14" line 24

    6. The "write-for-restart" line 26

    7. The "relaxation-param"line 27

    8. The "block-comment" line 28

    9. The "dimension" line 28

    10. The "handedness" line 29

PRECEDING PAGE BLANK NOT FILMED

PAGE ii INTENTIONALLY BLANK

Page

11. The "polar-axis" line 30

12. The "face" line 31

13. The "norm/sect" line 34

14. The "lighten/tighten" line 36

15. The "lighten-at" line 38

16. The "tighten-at" line 38

17. The "read-in-fixed" line 39

18. The "plane-normal-to" lines 40

19. The "cylinder-about" lines 42

20. The "ellipsoid" line 44

21. The "collapsed-to-an-axis" lines 45

22. The "collapsed-to-a-point" lines 47

23. The "match-to-face" lines 49

C. File11 – body definition arrays 50

D. File16 – control scalars for re-start 51

V. Running 3DGRAPE and evaluating its output 56

A. A "game plan" for running it 56

B. Reading the printout 57

C. What to do if it blows up 59

VI. Example 1: Grid about a simulated helicopter fuselage 61

VII. Example 2: C-O type grid about a wing on a wall 66

VIII. Example 3: H-H type grid about a wing on a wall 72

Page

IX. Theoretical development 75

References 80

Appendix A: File10 input data for Example 1 81

Appendix B: Program which makes file11 data for Example 1 84

Appendix C: File16 input data for Example 1 87

Appendix D: File10 input data for Example 2 90

Appendix E: Program which makes file11 data for Example 2 93

Appendix F: File16 input data for Example 2 98

Appendix G: File10 input data for first run of Example 3 101

Appendix H: Program for file11 for first run of Example 3 105

Appendix I: File10 input data for second run of Example 3 111

Appendix J: Program for file11 for second run of Example 3 113

ABSTRACT

This document is a users' manual for a new three-dimensional grid generator called 3DGRAPE. The program, written in FORTRAN, is capable of making zonal (blocked) computational grids in or about almost any shape. Grids are generated by the solution of Poisson's differential equations in three dimensions. The program automatically finds its own values for inhomogeneous terms which give near-orthogonality and controlled grid cell height at boundaries. Grids generated by 3DGRAPE have been applied to both viscous and inviscid aerodynamic problems, and to problems in other fluid-dynamic areas. The smoothness for which elliptic methods are known is seen here, including smoothness across zonal boundaries.

An introduction giving the history, motivation, capabilities, and philosophy of 3DGRAPE is presented first. Then follows a chapter on the program itself. The input is then described in detail. A chapter on reading the output and debugging follows. Three examples are then described, including sample input data and plots of output. Last is a chapter on the theoretical development of the method.

I. INTRODUCTION

In 1977 and 1978 J. L. Steger and this author researched the generation of two-dimensional (2-D) grids for airfoils by the solution of elliptic partial differential equations (PDE) (ref. 1). This approach was pioneered principally by J. F. Thompson in the early 1970s (refs. 2 and 3). That work was so significant that the entire approach of using elliptic PDEs to generate grids is sometimes referred to generically as "Thompson's method." But the crux of the matter is in the choice of right-hand-side (RHS) or inhomogeneous terms. It is the effect of those terms in the Poisson equations which give the user the ability to "push and pull" points around, to control the grid, and to tailor it to particular requirements. Thompson's RHS terms gave the user a great deal of control, but they were rather complicated. They consisted of nested summations of terms for each point, requiring the user to supply values for many free parameters with little guidance.

When this situation was surveyed, a need was seen for a variation on Thompson's terms which retained as much as possible of that method's ability to control the grid, but was simpler to use. It was reasoned that the boundaries of the grid were the most critical regions; it was there that the flow-solvers typically encountered their highest gradients. It seemed that if the grid could be made to behave nicely in the neighborhood of its boundaries, and be smooth in its interior, then that would constitute a reasonable minimum of grid control and a reasonably good grid would result. It was also hoped that this simpler control could be achieved using simpler RHS terms, defined fundamentally only along the boundaries, and having their influence decay in some simple fashion with distance away from those boundaries toward the interior.

The first step was to add geometric constraints to the problem, defining "well behaved at a boundary." The constraints added were (1) that the grid lines intersecting the boundary do so at a user-specified angle, typically 90°, and (2) that the distance along those lines, between the boundary and the first node in the field, be directly specified by the user. From a mathematical point of view, this was equivalent to adding two new equations, the elementary mathematical statements of the two geometrical constraints, to the Poisson equations which defined the grid. It was hoped that those two new equations would somehow yield a solution for the RHS terms, which could be considered as two new unknowns. That linkage between the two new equations and the RHS terms was discovered. It works well, although its mathematical derivation is not obvious.

The result of this was a new algorithm (ref. 4) for generating grids by the solution of Poisson's differential equations. This method has RHS terms which are simple, and are found automatically by the algorithm as the numerical grid generation solution proceeds. The only input the user must give for the RHS terms is a simple specification of (1) the angle with which lines are to intersect the boundary, and (2) the desired distance out to the first node in the field. The resulting grid obeys those constraints, to the extent that the solution of the difference equations approximates the solution to the differential equations. The grid gives near-orthogonality at the boundaries, user-specified cell height at the boundaries, and smoothness in the interior of the grid.

As of 1979 this grid-generation technology, in two dimensions, existed only as a rough research-type program, coded for a limited collection of hard-wired cases. This author was urged to repackage the technology in a neat, robust, widely applicable, well-documented, user-friendly, export-quality program. It was completely recoded in this way, and the GRAPE airfoil grid generation program (ref. 5), now sometimes referred to as 2DGRAPE, was the result. It has been quite successful as an export program. Approximately 150 copies of it have been distributed by this author and COSMIC, and informal communications suggest that there have been many subsequent transfers. It is this author's belief that the "clean sheet of paper" recoding effort for export was a significant factor in the success of 2DGRAPE.

Work began almost immediately on the obvious extension of 2DGRAPE to three dimensions (3-D). The mathematical extension of the equations to 3-D was straightforward. Grids were generated, and flows were modeled, from simple ellipsoidal wing shapes (ref. 6) to realistic F-16 wing-bodies (refs. 7 and 8). That being the case, one might wonder why it has taken so long for 3DGRAPE (ref. 9) to emerge. There are two answers to that. The second is the time required by the recoding for export. But the first reason for the delay was the problem of topological complexity. 2DGRAPE was limited to topologies into which a simple rectangular computational domain could be warped. The 3-D analogy to that would be requiring that a single "computational cube" could be warped into the physical domain. ("Rectangular solid computational domain" might be more rigorous than "computational cube," but much less euphonic.) In 3-D generally, in real-world computational fluid dynamics (CFD) problems such as flow about an airplane or inside a turbine, one usually cannot warp one cube into the physical domain.

There are two solutions to this problem. One is to put custom modifications into the program for each different application, such as would be required to place a wing in a slit. But the complication of that approach increases rapidly, resulting in the code being very difficult to extend and maintain, and in the computational domain being more complicated than a simple cube. The other approach is to use zones. (The terms zone and block are used herein interchangeably). In a zonal approach, the physical domain is divided into regions, each of which maps into its own computational cube. It is believed that even the most complicated physical region can be divided into zones, with it being possible to warp a cube into each zone. This is especially true if pathological cases are allowed wherein a face of a zone can degenerate into a line (giving a zone which is wedge-shaped) or into a point (giving a zone which is pyramidal). So a grid generator which is oriented to zones, allowing communication across zonal boundaries where appropriate, solves the problem of topological complexity. 3DGRAPE is such a grid generator. The acronym is "Three-Dimensional GRids about Anything by Poisson's Equation."

The 3DGRAPE code makes no attempt to fit given body shapes and redistribute points thereon. Body-fitting is a formidable problem in itself. The user must either be working with some simple analytical body shape, upon which a simple analytical distribution can be easily effected, or must have available some sophisticated stand-alone body-fitting software. 3DGRAPE expects to read in already-distributed x,y,z coordinates on the bodies of interest, coordinates which will remain fixed during the entire grid-generation process.

3DGRAPE does not require the user to supply the block-to-block boundaries—either the shapes or the distribution of points thereon. 3DGRAPE will typically supply those block-to-block boundaries, simply as surfaces in the elliptic grid. Thus at block-to-block boundaries the following conditions are obtained: (1) grid lines will match up as they approach the block-to-block boundary from either side, (2) grid lines will cross the boundary with no slope discontinuity, (3) the spacing of points along the lines piercing the boundary will be continuous, (4) the shape of the boundary will be consistent with the surrounding grid, and (5) the distribution of points on the boundary will be reasonable in view of the surrounding grid.

This grid generator is a low-level tool. It offers a powerful building-block approach to complex 3-D grid generation. Users may build each face of each block as they wish, from a wide variety of resources. Thus an entire complex 3-D grid is constructed as the knowledgeable user has envisioned it. This building-block approach is the antithesis of the "turnkey" approach, wherein the user takes a hands-off stance and expects the program to do all of the thinking. The turnkey approach was rejected for two reasons, the first of which is that writing such a piece of software was beyond the scope of this effort. Secondly, with a turnkey system users frequently finds themselves wanting to make the system do a job outside the domain of problems it is designed to handle. Thus much effort can be spent trying to "trick" such a system into doing a problem it was not designed to do, with usually mediocre results.

There are several features 3DGRAPE lacks. It uses point-sucessive-over-relaxation (point-SOR) to solve the Poisson equations. This method is slow, although it does vectorize nicely. 3DGRAPE lacks interactive graphics, although any number of sophisticated graphics programs may be used on its stored output file. But one overriding passion consumed this author during the writing of 3DGRAPE, and that was versatility. The block structure, described in a subsequent chapter, allows a great latitude in the problems it can treat. As the acronym implies, this program should be able to handle just about any physical region into which a computational cube or cubes can be warped.

ACKNOWLEDGMENTS

The author gratefully acknowledges the contributions of all those who have used 3DGRAPE during its development process and who have contributed feedback which led to improvements in the finished product. The author is grateful to Joe Thompson for his pioneering work in elliptic grid generation, and for his gracious and encouraging example. The author will remain indebted to Joe Steger for his creativity and leadership in the development of the foregoing 2-DGRAPE program.

II. THE PROGRAM: ITS AVAILABILITY AND INSTALLATION

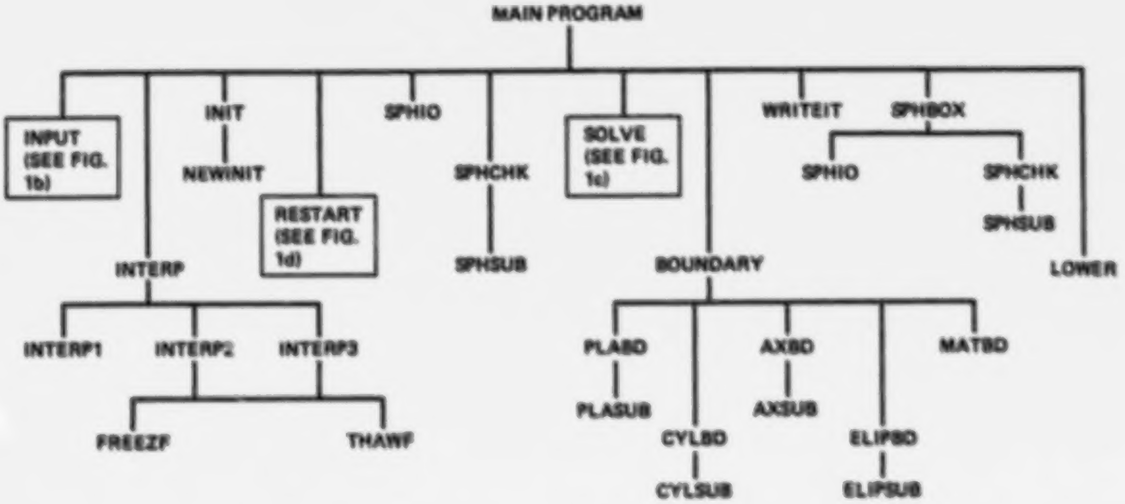
As of this writing it is intended that 3DGRAPE be distributed by NASA's clearinghouse for computer programs, the Computer Software Management and Information Center (COSMIC):

COSMIC
University of Georgia
382 East Broad Street
Athens, GA 30602
Phone (404) 542-3265

The detailed attributes of the distribution tape would have to be obtained from COSMIC, but the tape will consist of straightforward character data (not binary data). It is expected that the tape will have two files on it.

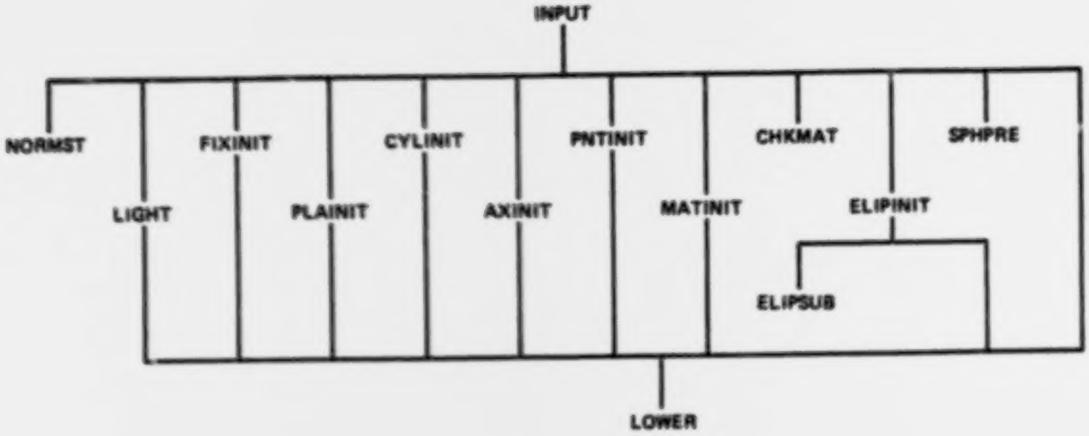
A. TAPE FILE 1: THE 3DGRAPE PROGRAM

The first tape file will be the FORTRAN source code for 3DGRAPE. 3DGRAPE consists of 14,043 lines of FORTRAN, containing 9,402 individual statements. The main program is first. Following it are 51 subroutines and one function subprogram, arranged in alphabetical order. Figure 1 is a chart showing how the subroutines are called.

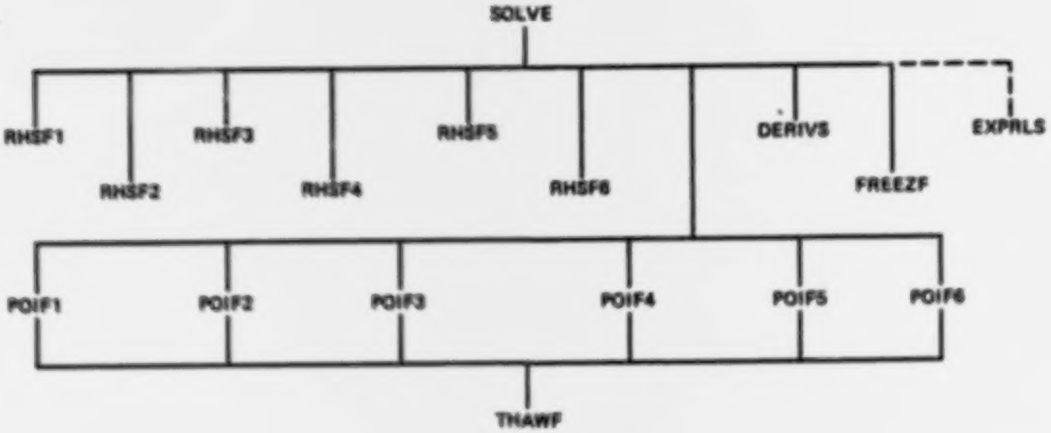


(a) Subroutines called by the main program, what they call, etc.

Figure 1.– Subroutine call chart for 3DGRAPE.

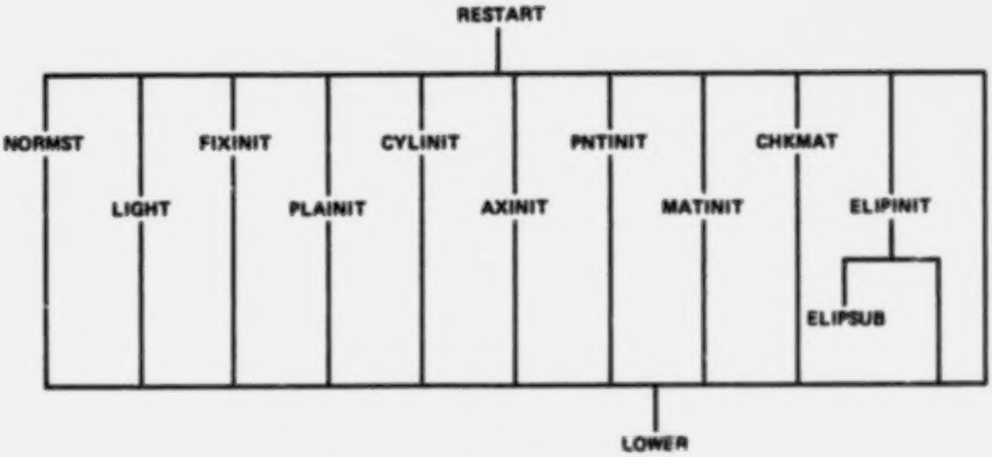


(b) Subroutines called by subroutine input, what they call, etc.



(c) Subroutines called by subroutine solve, what they call, etc. Subroutine exprls called only when run on IRIS workstation.

Figure 1.– Continued



(d) Subroutines called by subroutine restart, what they call, etc.

Figure 1.– Concluded

A true FORTRAN-77 compiler must be used to compile the program. This point is stressed, since there is at least one compiler in wide use today not does not meet the FORTRAN-77 standard: the CFT compiler on the CRAY-2. One of the syntaxes that compiler is unable to process is character substring manipulations, of which 3DGRAPE does many. 3DGRAPE does run successfully on the CRAY-2 using its CFT77 compiler.

The version of 3DGRAPE being distributed is the one which runs on the CRAY-2. To date it has also run on a CRAY X/MP, a VAX, and a Silicon Graphics IRIS 2500T workstation. The following rather short list gives the modifications necessary to port it to those other computers:

X/MP: Remove all the "open" statements.

VAX: Change variable "nbits" in subroutine "input" to 32 indicating that the VAX has 32-bit floating-point words.

IRIS: (1) Change variable "nbits" in subroutine "input" to 32.

(2)* Change all references to the "exp" function in subroutine "solve" to "exprls."

Since portability was kept firmly in mind while writing 3DGRAPE, porting it should be easy.

---

*The "exp" function supplied by the FORTRAN compiler on the IRIS, when linked for use with the floating-point accelerator board, gives wrong answers.

The coding is structured to the greatest extent possible. If-blocks and do-loops are indented. All variables are defined before use; no memory preset is assumed. The program does not assume that local variables in subroutines remain defined between calls. The "inner loop" in subroutine "solve" vectorizes on the CRAY-2 and CRAY X/MP without any compiler directives. Single-dimension addressing is used in the principal arrays, reducing wasted storage to almost nil.

The program has 11 common blocks, all of which appear in the main program. A parameter statement appears wherever common blocks are found which contains parameters specifying the dimension sizes. Thus the dimension sizes in the code are easily modified; the user should simply make a global change of the parameter definition. The individual parameters, their values in the code as it is supplied, and their meanings are listed below:

Parameter:	Supplied value:	Meaning:
limpts	200,000	Limit on the total number of points in the grid,summed over all blocks.
limsrf	25,000	Limit on the number of points allowed on all six surfaces of any one block.
limpqr	50,000	Limit on the number of surface points at which control is actually exercised, summed over all blocks.
limblk	20	Limit on the number of blocks.
limvec	300	Limit on the maximum value of the first index, j, for any block. Subroutines "solve" and "deriv" vectorize in this direction.
limhis	999	Limit on the number of iterations which can be stored in the convergence history arrays.

The storage requirements for the code are, of course, critically dependent upon the values chosen for these parameters. By inspecting the common blocks, it appears that the approximate storage requirements for a nominal N x N x N grid are

$$3N^3 + 159N^2$$

But that formula is only an approximation, and its value is frequently low. To that figure must be added space for other storage arrays, for the compiled and linked code, for input/output (I/O) buffers, etc.

As an example, the code using the supplied values for the parameters would just accommodate a 58 x 58 x 58 grid. The above formula with N=58 yields a storage requirement of 1,120,212 words. But the program thus dimensioned will just fit into a 2-million-word partition on the CRAY X/MP. So in this example a 200,000-point grid requires 2,000,000 words, for a ratio of 10 words of memory per point.

Following is a list of logical unit numbers used for input and output in 3DGRAPE, whether they are used for input or output, a statement of where they are used (main program or subroutine name), and a brief statement of what they are used for:

Unit no:	Input/ output:	Where used:	What for:
*	input	main program, input, restart	first two lines of input
*	output	everywhere	main printed output
10	input	input, light, plainit, cylinit, axinit, pntinit, matinit, elipinit	control scalars for new start
11	input	fixinit	x,y,z coordinates for fixed surfaces
12	output	fixinit	debug writes of data read from unit 11
14	output	writeit	main output of finished grid
15	output	restart	restart file
16	input	restart, light, plainit, cylinit, axinit, pntinit, matinit, elipinit	control scalars for restart
17	input	restart	restart file

It is most strongly recommended that the users of 3DGRAPE have at their disposal a state-of-the-art interactive graphics capability. It would be practically impossible to debug and evaluate a complicated grid without the ability to plot surfaces of choice; color them arbitrarily; and do rotations, translations, and zooming operations on them. 3DGRAPE was developed at NASA Ames Research Center using a Silicon Graphics IRIS 2500T workstation, with custom graphics software called 3DECANT written for the purpose by this author. The PLOT3D software package, also written at Ames for the IRIS, is adequate for the purpose. Other powerful scientific workstations and software packages are available about which this author cannot knowledgeably comment.

B. TAPE FILE 2: INPUT DATA FOR EXAMPLE CASES

It is expected that the second file on the tape will be exactly the same as the appendices to this manual. Those appendices consist of some of the input data files for the three example cases and simple FORTRAN programs which generate the remainder of the input data files for those example cases. Users who wish to run the example cases can separate the data and programs in this second tape file using an editor. They can run the input data generation programs, obtain the remainder of the input data files, and then run 3DGRAPE's three example cases.

Conceivably, all input data files for the three example cases could have been given in the appendices and reproduced on the second tape file. But for two reasons programs which generated some of those data files are supplied instead. First, data consisting of the x,y,z coordinates of points on the given surfaces tend to be voluminous and not very interesting, since for the most part they are page after page of floating-point numbers. Second, users of 3DGRAPE tend to write programs to generate such data files; no one (to this author's knowledge) has ever typed one in. Seeing such a program is more instructive than pouring over its numerical output. There is nothing remarkable about these small programs. They should compile and run easily on any machine.

III. ZONING

A. A PRIMER ON ZONING

Chapter I introduced the concept of breaking up the physical domain into zones. As computational fluid dynamics matures, it is being required to solve problems which are more and more of the "real world." No longer do we solve for flow about ellipsoids. Instead, we are asked so solve for the flow about such configurations as modern fighter aircraft with inlets, wingtip missiles, deflected control surfaces, and external stores. It is frequently impossible to map such a complicated physical domain into one computational cube. Instead, the physical domain is broken up into several zones, each of which maps into its own computational cube. Just as the zones in physical space contact each other in some way, and fluid flow passes from one zone to another, so must the flow solver allow communication between zones in computational space.

The new user of 3DGRAPE should first become used to thinking of a physical domain mapping into a computational cube. Figure 2 is a crude attempt to animate the warping of a computational cube into a curved duct in physical space. Note the correspondence between the floor of the duct and the floor of the cube, between the left side of the duct and the left side of the cube, etc. Just as the Cartesian computational grid will have lines connecting opposing faces, such as the ceiling and the floor, so will the physical domain have such connecting lines. Note that the difference between the 3-D networks of grid lines in the two spaces is that in computational space the lines are orthogonal and uniformly distributed, whereas in physical space they are "bent" or "warped," and the spacing is not uniform. Faces map into faces, lines map into lines, and points map into points.

Figure 3 shows another example, this being a grid about a quadrant of a sphere (half a hemisphere). The succession of views is another attempt to show in several steps how a computational cube can be warped into some shape in the physical domain. A new complication here is that one of the faces in the computational domain is collapsed to the spherical axis, a line, in the physical domain.

Once one has become comfortable with the concept of mapping the physical domain, no matter how it is shaped, into a computational cube, the next step is the extension to multiple zones. Suppose, for example, one were attempting to make an H-H type grid about a wing. The first H in this terminology says that as one views a section of the grid taken normal to the span, showing the airfoil shape, one will see one family of grid lines running generally fore and aft and the other family of lines running generally up and down. Hence the wing resembles the cross-bar in a letter H. The second H in this terminology says that as one views a section of the grid taken normal to the freestream, showing a spanwise cut of the wing, one will see one family of grid lines running generally in the spanwise direction and the other family of lines running generally up and down. Hence the terminology H-H.

Barring the use of a "slit," which produces a computational domain more complicated than that of the basic cube, it is not possible to solve that gridding problem with one zone. One zone could treat the upper surface or the lower surface, but not both. Alternatively, one could wrap a one-zone grid around the wing, but the result would not be of the H-H type. The solution to this dilemma is to use two zones—one above the wing and one below (fig. 4). Each of the two zones in physical space maps into its own computational cube.

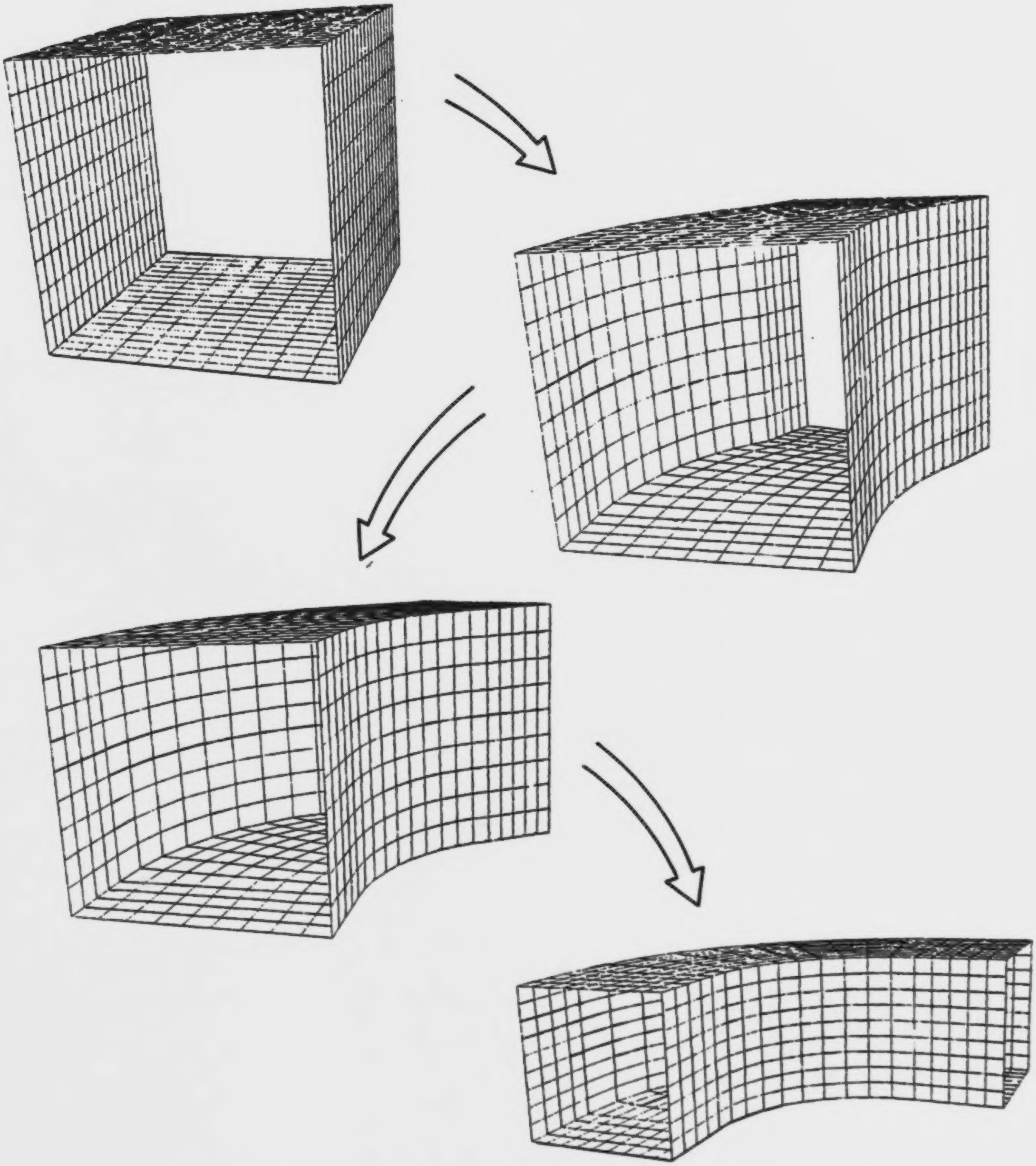


Figure 2.– Composite figure showing how a cube may warp into a curved duct.

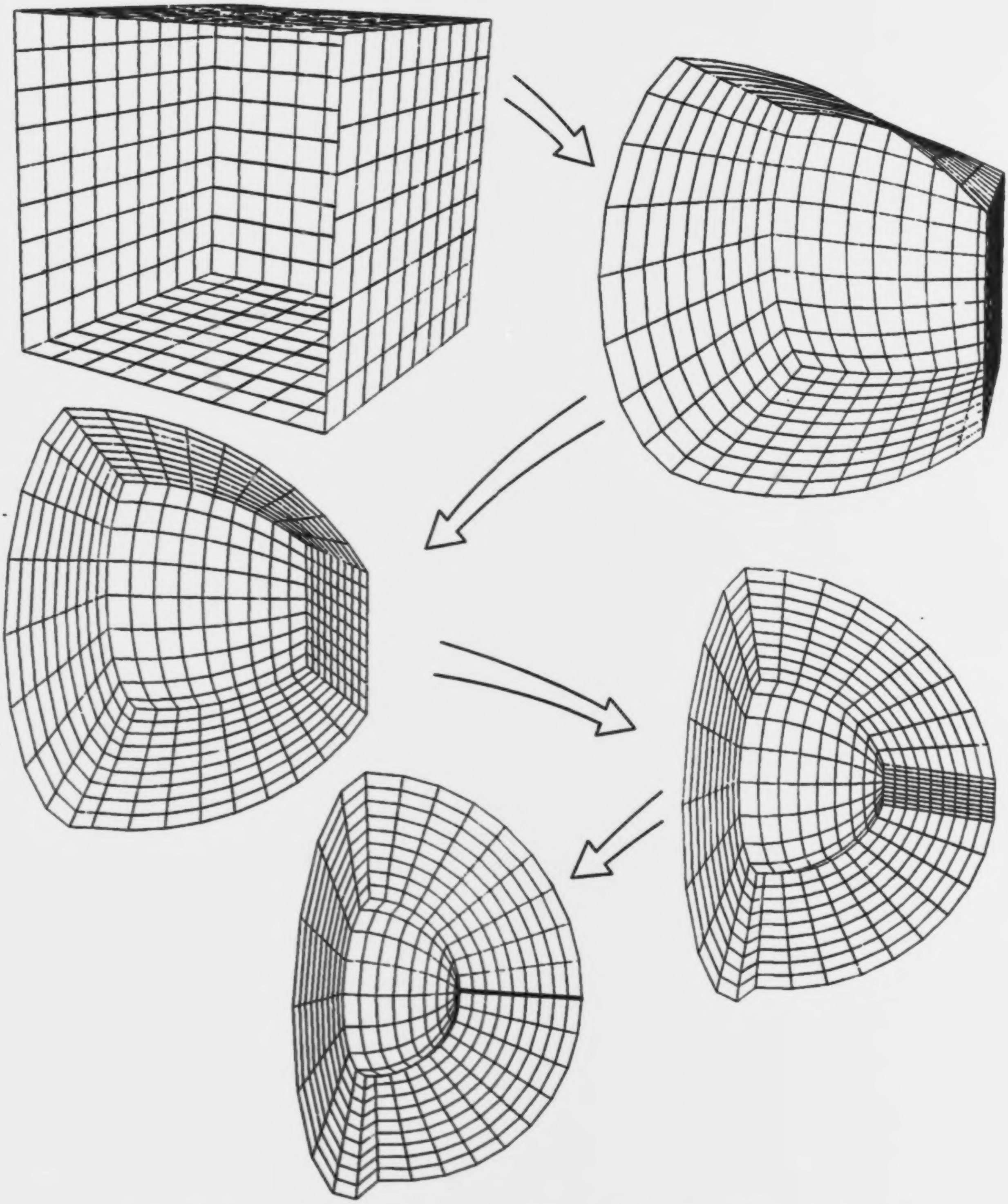
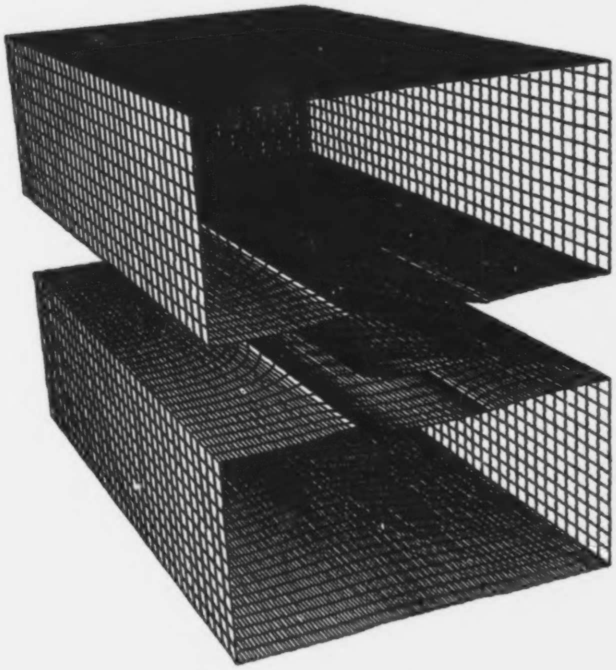
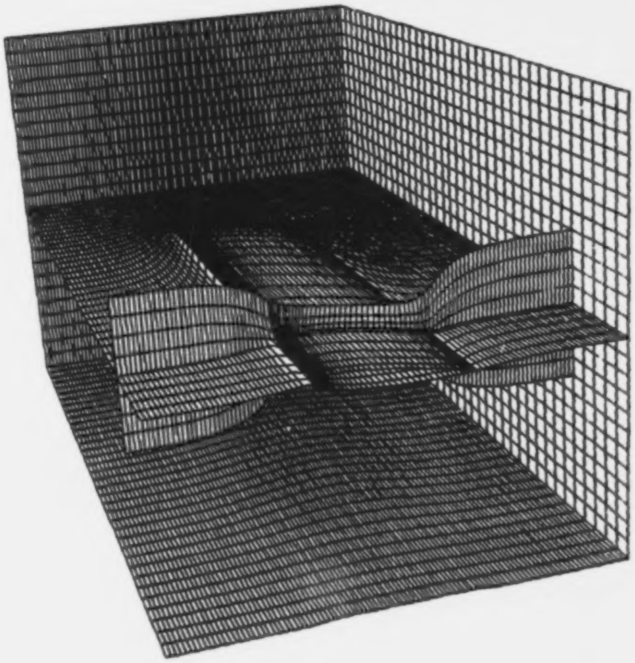


Figure 3.– Composite figure showing how a cube may warp into a spherical quadrant.

ORIGINAL PAGE IS
OF POOR QUALITY



(a) Expanded view of two blocks above and below wing.



(b) Selected surfaces in assembled two-block grid, seen from root end.

Figure 4.– H-H type grid about wing.

Complicated physical regions are gridded by the use of many zones, and flow is modeled in complicated physical regions by the use of many zones. But the zoning used in making the grid is frequently not the same zoning as is used in modeling the flow. Frequently the limitations of the grid generator impose a need for a particular zoning, and that zoning is found to be not appropriate for the flow solver. In such a case the grid is generated as many zones, and those zones are then copied together into a few zones (e.g., one zone). Those few zones are then divided again into a different set of many zones for use in the flow solver. That redivision is typically performed as a post-processing step by some simple application-specific interface program.

B. ZONING WITH 3DGRAPE

3DGRAPE has the ability to locate the points on the boundaries of the zones in a variety of ways. 3DGRAPE also has the ability to cause the points just inside the zones to be generated such that near-orthogonality and controlled grid cell height are imposed. But the user should keep firmly in mind that these are two distinctly different matters, and they are specified differently in the input to the code.

3DGRAPE offers seven different ways to locate the points on the boundary faces of the zones. They are

1. The x,y,z locations of points on boundaries may be read in from a data file and remain fixed for the entire grid-generation process. This boundary treatment is typically used for the "given shape," about which or inside of which the user wants to make a grid.

   Note that 3DGRAPE has no facility to redistribute the points on that given shape. 3DGRAPE expects to read in points which have already been distributed. Thus for all but simple analytically defined shapes, the user must have available a surface-fitting program.

   These x,y,z points on the boundary surface must be ordered with two running indices, and their distribution should be dense or sparse as is appropriate to the problem.

2. The points on the boundary may be constrained to lie on a plane normal to one of the Cartesian coordinate axes. The actual location of the points on that plane will be dictated by and consistent with the elliptic solution for the points inside of the block. The boundary conditions in 3DGRAPE are explicit, meaning that some location for the boundary points is first assumed, then points inside the block are relocated by taking one solution step, and then from that interior solution the boundary points are relocated by extrapolation. That entire process is iterated to convergence.

   Simple extrapolation to a plane, such as by passing a parabola through neighboring points, tends to be unstable and produce unacceptable results. So a more sophisticated extrapolation procedure is used. A parabola is passed through the three points in the interior nearest the boundary plane, labeled 1, 2, and 3 in figure 5. From that parabola the slope of the curve at point 1 is found. That first parabola is then discarded. Three conditions are known from which a new parabola is found.

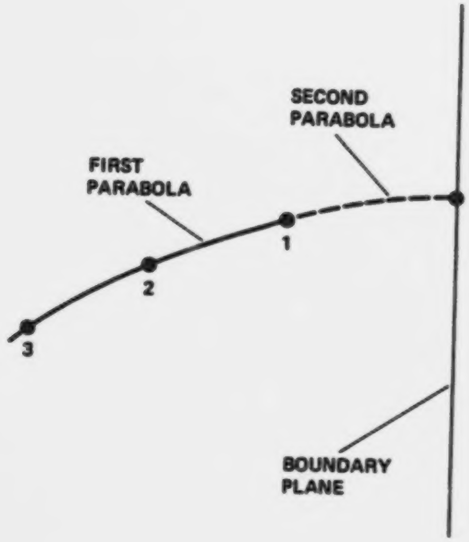


Figure 5.– Sketch showing extrapolation to planar boundary face.

They are (1) the location of point 1, (2) the slope of the curve at point 1, and (3) the slope of the curve at the boundary plane, such that the line is perpendicular to that boundary. That new parabola is evaluated at the boundary to get the location of the point on the boundary.

3. The points on the boundary may be constrained to lie on the surface of a cylinder (or a section of a cylinder) which has its axis coincident with one of the coordinate axes. Points on the cylinder will be extrapolated from points inside the block.

4. The points on the boundary may be constrained to lie on the surface of an ellipsoid which has its axes coincident with the coordinate axes. A sphere is a special case of an ellipsoid, and thus spherical boundaries are available. Points on the ellipsoid will be found by extrapolating from points inside the block.

5. The points on the boundary may be collapsed to a line, with that line being one of the coordinate axes. The location of the points on that line will be found by extrapolating from points in the interior of the block.

6. The points on a boundary may be collapsed to a point, located anywhere, and fixed for the entire grid-generation process.

7. The points on a boundary may abut the points on some other boundary. This is the facility which allows floating block-to-block boundaries. Two boundaries are specified, either of the same block or of different blocks. A solution step is taken to update the points inside the block(s), and then the points on the boundary are relocated by simply taking the two nearest interior points, one on each

side of the floating boundary, and finding the midpoint between them. Block-to-block boundaries are thus double-stored—once as part of each block.

Any of the above boundary treatments may be applied to each of the six faces of a block.

But real-world problems require even more versatility than that. Consider again the two-block grid in figure 4. The upper surface of the lower block touches the lower surface of the upper block in front of the wing, outboard of the wing, and behind the wing. But it also touches the lower surface of the wing itself. So which of the seven boundary treatments is appropriate for this upper face of the lower block? Obviously, none is appropriate by itself.

The solution to this dilemma is to divide that face of that block into sections. The lower face of the upper block is divided into sections similarly. Then the section in front of the wing is made to abut the corresponding section on the surface above it, the section outboard of the wing is made to abut the section above it, the section rearward of the wing is made to abut the section above it, and the section which touches the wing has its points read in and fixed. The abutting sections produce a floating block-to-block boundary, while the wing shape is preserved.

This need for dividing faces into sections is provided for in 3DGRAPE. Any face may be divided into sections. Any of the above list of boundary treatments may be applied to any section. The user should reread the above list, substituting "the section of the boundary" for "the boundary."

For purposes of keeping things straight, the faces of each zone are numbered from 1 to 6. By the nature of the problem, one of the three indices must be fixed on each face. For a user-specified arrangement of the indices, the face numbers are hard-wired into 3DGRAPE. The following table gives those predefined face numbers:

Face no.:	Fixed index:	Fixed at what value?	Indices running over the face:
1	j	1	k,l
2	j	its maximum (dimension)	k,l
3	k	1	j,l
4	k	its maximum (dimension)	j,l
5	l	1	j,k
6	l	its maximum (dimension)	j,k

Thus the index j runs from 1 to its maximum between faces 1 and 2. Therefore faces 1 and 2 must be located somehow "opposite" one another. Similarly, face 3 must oppose face 4, and index k runs between them. Face 5 must be opposite face 6, and index l runs between them. In figure 3 the spherical quadrant, i.e., the "inner boundary," is face 3, the outer boundary is face 4, and the index k runs between them. These predefinitions do not in any way constrain the cases which 3DGRAPE can treat nor how the indices may run. The only constraint is on what number will be attached to what face.

It was stated earlier that locating the points on the boundaries of the blocks is a distinctly different matter from obtaining orthogonality and control of grid cell height within the blocks near the boundaries. These latter capabilities are illustrated in figure 6. Right-hand-side terms have been added to the governing grid generation equations which cause the grid to have these qualities. Actual values for those terms are found iteratively as the grid-generation solution proceeds. All the user need supply is the desired value for the height of the cells on the boundary; the program does the rest. This kind of control is available on all six faces of the zone. The control may be deactivated at any or all of those faces. The RHS terms include a decaying exponential factor which causes the magnitude of the terms, and thus their influence on the grid, to be reduced with distance from the boundary. Thus in the middle of the zone where all of the control terms have decayed to practically zero, as well as in the vicinity of any face where the control is turned off, the grid is locally a Laplacian grid, which is a very smooth grid.

Control should be used only on faces of zones which have boundary points located by treatment no. 1, where the points are read in and remain unchanged for all computational time. Clustering points to a floating boundary can produce instability in the grid-generation solution process.

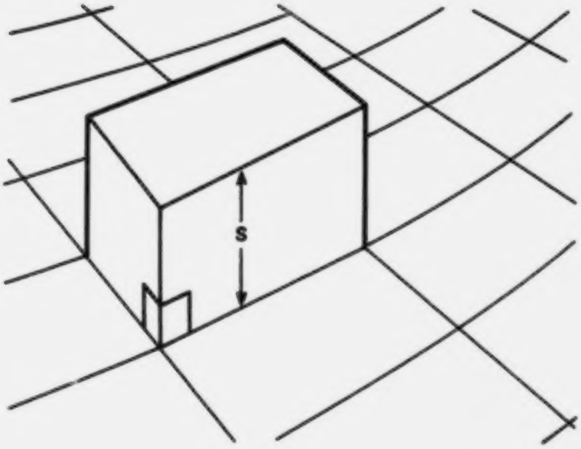


Figure 6.– Sketch of grid cell on boundary surface–3DGRAPE's control terms impose near-orthogonality and control of grid cell height(s).

Control is active or not on each face of each zone. As the program is presently coded, it is not possible to activate or deactivate the control by sections. This can increase the complication of the zoning for some problems. For example, consider the H-H type grid about the wing in figure 4. That grid, as shown, cannot actually be generated by 3DGRAPE using the two-block topology as indicated. That grid has control on the upper and lower wing surfaces, but it does not have control on the remainder of the planform surface, in which the wing resides. So generating that grid, using a two-block topology with the planform surface being the block-to-block boundary, would require control to be active on one section of that surface (the wing) and inactive on other sections (in front of the wing, behind the wing, and outboard of the wing). That is not possible with 3DGRAPE. That grid was actually generated using a more complicated topology with five blocks.

The problem of H-type grids for wings requires further comment. A problem arises in the distribution of points on the planform surface itself. The grid spacing in the streamwise direction is fine on the wing near its leading and trailing edges, but it can suddenly become quite coarse as one moves off of the wing in either the upstream or downstream direction. The only solution to that problem found by this author is to supply a properly clustered surface grid for the entire planform surface, not just the wing, as input to 3DGRAPE. In that case, the entire planform surface would be read in and fixed, as is the wing. Thus a reasonable distribution of points on that surface can be supplied and preserved. That properly clustered planform surface grid could be supplied by a general-purpose, surface-gridding program.

Another solution to the problem of supplying the planform surface grid, awkward but workable, is to use 3DGRAPE twice. This is illustrated in Example case 3. The perimeter of the wing, a line consisting of the leading edge, the tip, and the trailing edge, is identified. Several (typically five) copies of that perimeter are stacked one above another by adding or subtracting constant values of the vertical coordinate. That perimeter, so replicated, becomes a vertical "wall." That wall is then considered as a separate 3-D grid-generation problem. Points can be attracted to that wall by activating control thereon using 3DGRAPE. One of the horizontal surfaces from the finished grid is extracted and is used as the fixed planform surface.

Because 3DGRAPE extrapolates to most boundaries, high curvature should be avoided near corners. Imagine a corner between a "floor" and a "side wall." Consider extrapolating horizontally to points on that side wall from various interior points. One can imagine that that process would work well, assuming that the floor is flat. But what if the floor is highly curved near the corner? In that case, the resulting distribution of points on the side wall might be uneven, or might even have lines crossing. Thus block boundaries should not be placed in regions having high curvature in the fixed boundaries.

IV. INPUT

Once the users have their intended zoning firmly in mind, they are ready to prepare the input. 3DGRAPE reads its first two lines of input from the terminal and the rest of its input from stored files.

A. THE FIRST TWO LINES

The first thing 3DGRAPE does as it begins execution is to write a prompt asking whether this run is a new start or a restart. The proper response is to type in one of two character strings, either "newstart" or "re-start."

The user is then prompted for the name of the file which will be used for input on file10 in the case of a new start, or on file16 in the case of a restart. Just a carriage return, preceded by no characters, will cause the default filename "file10" to be used in the new-start case, and "file16" to be used in the restart case.

The preceding discussion of the first two lines of input assumes that 3DGRAPE is being run on an interactive machine. If it is being run on a batch machine, the prompts will be written to the printout file, along with an echo of the input. The actual input of these two lines in this case will come from the main job input stream. Literally, they are read by the logical unit denoted in the program by an asterisk, as in "read(*,100)...."

The user should realize further that for most batch machines, such as the CRAY X/MP, the installation of the program will require removal of all "open" statements from the code. In those cases, all filenames read from the input will be ignored (with the partial exception of unit 12, see below). When the program is installed without open statements, the linking of the unit numbers and the data files will be done by job control language (JCL). So in those cases the second datum read, a file name, will be ignored.

B. FILE10 – CONTROL SCALARS FOR NEW START

Input on file10 is formatted, and thus is human-readable. All data for file10 must be in exactly the right columns. Those column numbers will be clearly delineated below, and they must be followed exactly. There is some consistency here: face numbers will always be read in I1 format, block numbers in I2 format, indices in I3 format, floating-point numbers in F12 format, and file names in A15 format. Character strings may be entered in either upper or lower case (or even a mixture of the two), with the exception of file names. If the user's operating system is case-sensitive, as is UNIX, then the file names must appear just as they are to be used.

There are places in the input where the user is given the option of entering either a character string or a floating-point number. The program is smart enough to sort out that form of input. It was stated earlier that floating-point numbers are read in F12 fields. To be precise, the format specification in 3DGRAPE is F12.0. But that does not mean that only whole numbers may be read. According to the rules of FORTRAN, a decimal point in an input record overrides any placement of the decimal point implied by the format statement. Thus the user may put a decimal point anywhere in the floating-point input numbers.

For the sake of experienced users looking for a quick reference, the discussion of each input will be preceded by a table giving all relevant data. Note that some inputs require continuation lines. Reading down the table will be a list of the line number (for inputs with continuation lines) and the different fields on the line(s). Reading across will be first the range of column numbers for that field. Then a letter will indicate what type of datum this is: "k" for keyword (a character string which must be entered exactly as stated, and which is required for readability), "i" for integer, "f" for floating-point number, "n" for file name, or "c" for a character string. In some places the user may put into a field either a character string or a floating-point number; in that case the datum type will be given by "c/f." In other places the user may put into a field either a character string or an integer; in that case the datum type will be given by "c/i." To the right of that will appear a brief description of what that datum is. The table will be followed by an example, taken from the first example case wherever possible. Immediately below that will be a column number key. After that will follow a discussion of the indicated input line.

The input 3DGRAPE expects to read from file10 is shown schematically in figure 7. It begins with several lines which give information about the entire grid and about the entire run of 3DGRAPE. It then goes into an outer loop on block number, and for each pass it reads information about the block. Inside that is an intermediate loop on face number and for each pass it reads information about the face. Inside that is an inner loop on section number within the face, reading information about each section. At the conclusion of those nested loops, it is finished reading from file10.

1. The "run-comment" lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-20	k	"run-comment "
	2	21-70	c	free-field comment describing this run

```
run-comment         Example:  hemisphere-cylinder-cone
run-comment         simulation of helicopter fuselage.
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The file10 input begins with exactly two of these lines. The comments on them will annotate the printout file, and they will help the user to remember what each file10 dataset was used for.

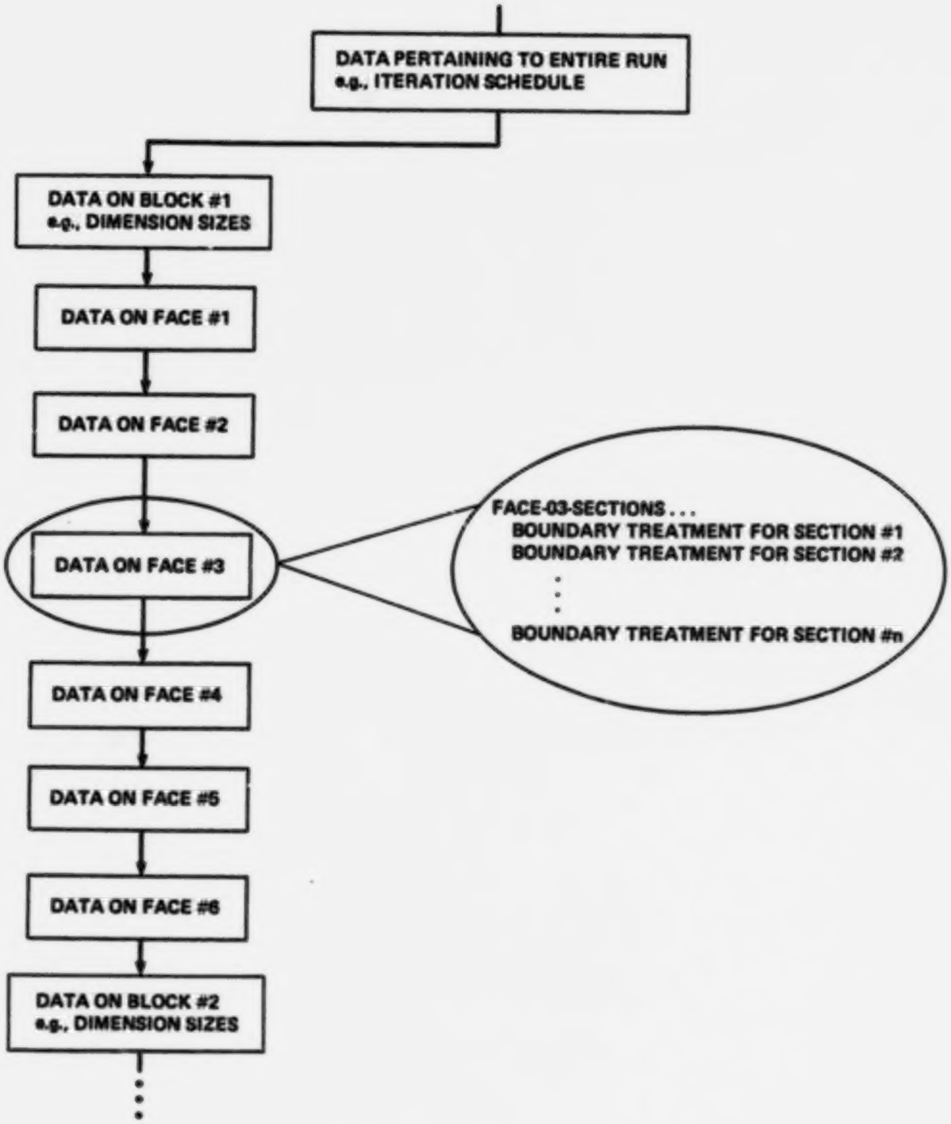


Figure 7.– Schematic summary of file10 input.

2. The "number-of-blocks" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-17	k	"number-of-blocks="

	2	18-19	i	number of blocks in this grid
	3	20-58	k	"-number-of-parts-in-iteration-schedule="
	4	59-60	i	number of parts in this iteration schedule

```
number-of-blocks=03-number-of-parts-in-iteration-schedule=03
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The iterations which 3DGRAPE will perform on this run are divided into parts, with varying characteristics for each part. The maximum number of parts is 10.

3. The "iterations" lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-11	k	"iterations="
	2	12-14	i	the number of iterations in this part
	3	15-23	k	"-control="
	4	24-25	c	global switch on control, either "ye" or "no"
	5	26-38	k	"-coarse/fine="
	6	39-44	c	"coarse" or "fine "

```
iterations=020-control=no-coarse/fine=coarse
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

One of these lines will be read for each part in the iteration schedule, defining that part and its characteristics.

It will be seen later how one goes about activating or deactivating the control terms on each face of each block. But the character string in columns 24-25 on this input line is a global switch which overrides all face-by-face specifications. String "ye" allows face-by-face invocation of control for this part of the iteration schedule; "no" turns control off at all faces for this part.

A procedure to speed up convergence has been added to 3DGRAPE. It starts with a very coarse grid, consisting of every third point in each of the three coordinate directions. In one part of the iteration schedule, this coarse grid is iterated to convergence, including the RHS terms. That coarse solution is then interpolated to cover all grid points in every direction. Then another part in the iteration schedule follows wherein another iteration to convergence takes place, using the interpolated grid as initial conditions. The first iteration goes fast because it does approximately 1/27th as much arithmetic per step as it would otherwise. The second iteration goes fast because it starts with initial conditions which are very close to the final solution.

The effectiveness of this technique varies greatly from case to case, but the user can count on a reduction in CPU time of at least 50%, sometimes much more. There is a drawback, and it is that the number of points in each of the three coordinate directions in every block must be of the form 3n+1 for n some integer greater than or equal to 4. In some cases this requirement is found to be burdensome, and use of this speedup procedure is not possible. The "coarse" or "fine " in columns 39-44 indicate whether this part of the iteration schedule is to be coarse or fine. Any number of coarse steps may be followed by any number of fine steps, but no coarse step may follow a fine step.

4. The "filename-11" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-18	k	"filename-11-input="
	2	19-33	n	name of file for input as file11
	3	34-53	k	"-filename-12-output="
	4	54-68	n	name of file for debugging output on file12

```
filename-11-input=file11ex1      -filename-12-output=
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

File11 is described in detail in the following section. The name of that file is found in columns 19-33 on this line. Remember that when 3DGRAPE is installed without open statements, as is typical on batch machines, this and all other file names (with the exception of file12, described immediately below) are ignored.

Columns 54-68 contain the name of the file to receive debugging output on unit 12. That file name serves a dual purpose. First, its presence or absence serves as a switch telling 3DGRAPE whether to write or not write (respectively) that data. Its second purpose, when 3DGRAPE is installed with open statements, is to provide the name of the file which will receive that debugging output. When 3DGRAPE is installed without open statements, the name of the file is supplied by JCL, and what appears in columns 54-68 is just a switch. This output is for debugging the input from file11. It is voluminous, and its output is not recommended unless the user is desperate and has no graphical debugging aids.

5. The "filename-14" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-24	k	"filename-14-grid-output="
	2	25-39	n	filename for main grid output
	3	40-45	k	"-form="
	4	46-52	c	"3dgrape" or "plot3d" or "charact"

```
filename-14-grid-output=ex1.bin        -form=3dgrape
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The main grid output may take any one of three different forms, as specified in columns 46-52. The most compact way of describing those forms is to use the FORTRAN language, rather than to use English. The reader's ultimate purpose in reading this section is to enable him to prepare read statements for the grid file; that can be most expediently done by seeing the write statements which created it.

The code below is not literally excerpted from 3DGRAPE. It differs in data structure and variable names. However, this code would produce identical results and is easy readable. In the code, maxblk is the number of blocks. The array element jmaxa(nblk) is the maximum value of the first subscript for block number nblk, and similarly the second and third subscripts k and l. The x coordinate is assumed to be stored as x(j,k,l,nblk), and similarly y and z.

If "3dgrape" is specified, the data on file14 are written in a form which this author believes to be the simplest and most straightforward. It is a form created by this author for this program, but is easily adaptable to other uses. It is identical to that which would be produced by the following simulated code:

```
open(unit=14,status='new',form='binary',file='ex1.bin')

write(14) maxblk

do 1 nblk=1,maxblk
```

```
         jmax=jmaxa(nblk)
         kmax=kmaxa(nblk)
         lmax=lmaxa(nblk)

         write(14) jmax,kmax,lmax

         write(14) (((x(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     1             (((y(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     2             (((z(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax)

 1      continue

        close(unit=14)
```

If "plot3d" is specified, the data on file14 are written in the form required by the well-known NASA graphics program PLOT3D (refs. 10 and 11). If there is only one block, the data are written in PLOT3D's single-block format. If the number of blocks is greater than one, the data are written in PLOT3D's multiple-block format. The output is identical to that which would be produced by the following code:

```
        open(unit=14,status='new',form='binary',file='ex1.p3d')

        if(maxblk.gt.1)  write(14) maxblk

        write(14) (jmaxa(nblk),kmaxa(nblk),lmaxa(nblk),nblk=1,maxblk)

        do 1 nblk=1,maxblk

         jmax=jmaxa(nblk)
         kmax=kmaxa(nblk)
         lmax=lmaxa(nblk)

         write(14) (((x(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     1             (((y(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     2             (((z(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax)

 1      continue

        close(unit=14)
```

If "charact" is specified, the data on file14 are written in as formatted data, or ASCII character data. This is useful for users running on computers connected to a network which does not have the facility to transfer binary data. A main grid output file created this way will be several times as large as if either of the two other options had been used, and it will take several times as long to read, but for some users this approach is unavoidable. This form is essentially the "3dgrape" form converted to formatted output.

```
      open(unit=14,status='new',form='formatted',file='ex1.asc')

      write(14,100) maxblk
 100  format(3i10)

      do 1 nblk=1,maxblk

       jmax=jmaxa(nblk)
       kmax=kmaxa(nblk)
       lmax=lmaxa(nblk)

       write(14,100) jmax,kmax,lmax

       write(14,101) (((x(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     1               (((y(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     2               (((z(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax)
 101   format(5e15.6)

 1    continue

      close(unit=14)
```

6. The "write-for-restart" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-18	k	"write-for-restart="
	2	19-20	c	either "ye" or "no"
	3	21-40	k	"-filename-15-output="
	4	41-55	n	filename for restart file

```
write-for-restart=no-filename-15-output=restartex1
123456789012345678901234567890123456789012345678901234567890123456789O
000000000111111111122222222223333333333444444444455555555556666666666667
```

3DGRAPE has a restart capability. One can run it a while, plot the results, and then decide to run it some more, either with or without some changes. To make that possible, 3DGRAPE must write out a file containing all it needs to continue where it left off. File15 is that file. It is output by 3DGRAPE in the run antecedent to the restart, and then read back in on the restart run. It is a very large file, containing the contents of most of the common arrays, and some other material as well. This author sees no reason why the user would ever need to examine the contents of this file.

The character-string "ye" or "no" in columns 19-20 determines whether the file is written. If a restart file is to be written, and 3DGRAPE is installed with open statements, this file name must be given in columns 41-55. Otherwise, this file name is ignored.

7. The "relaxation-param" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-17	k	"relaxation-param="
	2	18-29	c/f	either "keep-default" or a value for omega

```
relaxation-param=keep-default
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

3DGRAPE uses point-SOR to solve the Poisson equations. In that method there is a relaxation parameter, commonly called omega, which determines whether the solution is being overrelaxed or underrelaxed. This parameter must be between zero and two. Increasing it makes the solution converge faster, at the possible expense of instability. Putting the character string "keep-default" in columns 18-29 invokes the default, which is 0.8. Putting a floating-point number in those columns causes that number to be used instead.

The foregoing input data records give information about the entire grid-generation operation being conducted by this run of 3DGRAPE. Following these lines the program goes into an outer loop on the block numbers. For each block a group of lines must then be encountered which give characteristics of the block.

8. The "block-comment" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-20	k	"block-01-comment"
	2	21-70	c	free-field comment describing this block

```
block-01-comment    Hemispherical Nose Cap
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The block statement, along with the face statement described below, are the two kinds of input lines preceding which blank lines may appear. Any number of blank lines may be placed before a block or face input line for the purpose of making the file10 input more readable. Blank lines anywhere else in file10 will be errors.

The comment in the comment field of the block statement will be used to annotate the printout. The printout, described in detail in a subsequent chapter, will include a convergence history for each block. Those histories will be labeled with the comments from the corresponding block statements.

9. The "dimension" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-12	k	"dimension-j="
	2	13-15	i	maximum value of first subscript j
	3	16-28	k	"-dimension-k="
	4	29-31	i	maximum value of second subscript k
	5	32-44	k	"-dimension-1="
	6	45-47	i	maximum value of third subscript 1

```
dimension-j=019-dimension-k=031-dimension-1=022
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The dimensions of each block are variable, and may be set by the user at execution time. The only such limitation which must be set at compile time is on the total number of points summed over all blocks, described in Chapter II Section A. The dimension sizes must in every case be at least 4. If "coarse" iteration steps are to be performed, then the dimension sizes must be of the form 3n+1 for n some integer greater than or equal to 4.

10. The "handedness" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-11	k	"handedness="
	2	12	c	either "r" or "l"
	3	13-22	k	"-initcond="
	4	23	c	either "j" or "k" or "l"
	5	24-33	k	"-cart/sph="
	6	34-42	c	either "Cartesian" or "spherical"

```
handedness-r-initcond=k-cart/sph=spherical
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The "handedness" of the grid—either right-handed or left-handed—can vary from block to block. For Laplacian grids it is irrelevant. But for grids with control activated, it is used to choose the sign of a square root in the computation of the RHS terms.

The handedness of a grid can be determined according to the right-hand rule, or in the following equivalent way. Choose any point (j,k,l). A unit vector in the ξ direction is a vector from that point to the point (j+1,k,l). Similarly, a unit vector in the η direction is a vector from (j,k,l) to (j,k+1,l). And a unit vector in the ζ direction is from (j,k,l) to (j,k,l+1).

The three vectors will be bound tail-to-tail-to-tail at the point (j,k,l). Imagine them defining the axes of a locally Cartesian ξ,η,ζ coordinate system. Imagine an ordinary screw, placed coincident with the zeta axis. Then imagine rotating some point on the head of that screw from the positive ξ axis to the positive η axis. If that rotation produces movement of the screw in the positive ζ direction, then the grid is right-handed. If that rotation produces movement in the negative ζ direction, then the grid is left-handed.

The character in column 12 should indicate that handedness: "r" for right-handed or "l" for left-handed. Users frequently make mistakes on this point, producing grids with grid lines repelled from the controlled faces rather than attracted. Rather than agonize analytically over this point, the user encountering such symptoms should simply reverse the handedness and try again.

In starting an execution of the grid generator, once points have been initialized in some way on all six faces of the block, the need arises to initialize the points inside the block. The user will choose some index, running between two opposing faces. The program will take corresponding points on the two faces and linearly interpolate between them to produce an initial distribution. The user chooses which index to use in column 23, and by so doing, chooses which pair of opposing faces to use.

It has been stated that 3DGRAPE should be able to make a grid in any region into which a cube or cubes can be warped. This is true, but for cases having spherical topology, i.e., having a spherical axis (such as in fig. 3), certain mathematical singularities occur and special measures must be taken. The coordinates in such zones are transformed from Cartesian coordinates (x,y,z) into spherical coordinates (ρ,θ,ϕ). An iteration is performed on the grid in that space. Then the outermost four shells (or cubic surfaces) are converted back to Cartesian coordinates. Boundary conditions are applied, and the surfaces are transformed back into spherical coordinates. This is iterated to convergence, and the entire block is transformed back into Cartesian coordinates before being written out.

To utilize this option in any block, the user should put "spherical" into columns 34-42. Otherwise, "cartesian" should be entered in those columns. The spherical axis must be coincident with one of the coordinate axes.

11. The "polar-axis" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-11	k	"polar-axis="
	2	12	c	either "x" or "y" or "z"
	3	13-19	k	"-along="
	4	20	c	either "j" or "k" or "l"
	5	21-28	k	"-around="
	6	29	c	either "j" or "k" or "l"

	7	30-37	k	"-center="
	8	38-49	f	location on polar axis of approx. spher. center

```
polar-axis=x-along=k-around=l-center=   100.
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

This line is read only if "spherical" appears on the preceding line. In that case, 3DGRAPE needs to know which axis is the polar axis. That datum is entered in column 12. The program then needs to know which index runs along that axis, entered in column 20, and which index runs around it, entered in column 29. In the spherical case neither the body nor the outer boundary need be exactly spherical, but they should be somewhat similar to a sphere, i.e., topologically equivalent to a sphere. Given that, it should be possible to locate an approximate center to that sphere. That center would, of course, lie on the spherical axis. The location of the approximate center is given by entering its location on the axis in columns 38-49.

This concludes the inputs which give characteristics of the block. At this point 3DGRAPE goes into an intermediate loop on face number. It expects to read information which applies to each face. Blank lines may appear before a "face" line.

12. The "face" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-5	k	"face-"
	2	6	i	face number
	3	7-16	k	"-sections="
	4	17-18	i	number of sections into which face is divided
	5	19-26	k	"-normal="
	6	27-38	c/f	"uncontrolled" or cell height or "n-i-stations"
	7	39-43	k	"-abc="
	8	44-55	c/f	"keep-default" or stretching parameter

9	56-68	k	"-light/tight="
10	69-70	c	"ye" or "no"

```
face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The face numbers should appear in numerical order, from one to six. For the purpose of locating the points on the face, it may be divided into sections. The maximum number of sections per face is 10.

The data in columns 27-38 require some explanation. There are three different forms acceptable here. The first is "uncontrolled." This means that the control terms are deactivated on this face and should be used for any boundary that is not a fixed boundary. The second form of input is to simply enter a floating-point number. This activates the control terms on this face. The program will try to make the cells touching this face be locally near-orthogonal, and will try to make them the height given in user units by the floating-point number.

The third form of input in columns 27-38 was listed as "n-i-stations." The use of quotes around that datum is questionable, since that character-string as is should never be used. In place of the "n" a number from 2 to 9 should be substituted. In place of the "i" an index ("j" or "k" or "l") should be substituted. For certain problems the user might require cell heights on a face which are controlled, but are not uniform. 3DGRAPE allows the specification of cell heights which are invariant with respect to one index but are varying as a piecewise continuous linear function of the other index. This form of input allows that. The piecewise continuous linear function is defined by giving the desired cell height at several values of the index, including its end points. The number in place of the "n" is the number of points at which a value for the cell height is to be given. The index substituted for the "i" is the index at values of which cell heights are to be given. For example, "4-k-stations" means that at k equals 1, at k equals its maximum value, and at two intermediate values of k, cell heights will be given. The required cell heights between those places will be found by linear interpolation.

When control on a face is activated, 3DGRAPE will attempt to make the grid cells immediately adjacent to that face be orthogonal and of the indicated height. With distance from the face, into the interior of the block, control of skewness and cell height decays. Thus in the middle of the block the grid is essentially uncontrolled. That decaying control allows the distance between points on lines normal to the face to increase in a quasi-exponential manner with distance from the face. But how fast does that control decay with distance inward? There is a parameter, called abc, which influences the rate of decay. The default value for that parameter is 0.45. The user may override that default by placing a floating-point number in columns 44-55. A larger number, such as 0.60 or 0.70, will cause the control to decay more rapidly, and will make the grid-generation convergence more stable. Decreasing that parameter to values such as 0.40 or 0.35 will cause the control to be propagated farther into the field, at the expense of decreasing the stability of the grid-generation convergence.

The foregoing discussion of the strength of the control assumes that it is uniform over the entire face. But there are occasions when the user wants for it to vary. For example, suppose the user is making a grid around the fuselage of an aircraft and there is a sharp strake edge protruding from the side of that fuselage. Elliptic grid generators carry a fundamental assumption that the boundaries are smooth, i.e., that the slope of the boundary is continuous. But at our hypothetical strake edge, the boundary is not smooth; the slope is discontinuous. If a surface grid line is allowed to go back along the fuselage, along the edge of that strake, it could be said that there is a surface grid line across which the boundary's slope is discontinuous. Continuing with the example, seen in figure 8, if the index j goes from front to back and l goes around, then it could be said that there is a value of l at which the slope in the l direction is discontinuous for all values of j. Since the elliptic grid generator assumes smooth boundaries, and there is a line on the boundary across which it is not smooth, special measures must be taken along that line. If this is not done, the grid generator will produce unacceptable results there or even fail to converge. The special measure which solves this problem is to average the RHS terms across that strake edge. That is, computed values of the RHS terms at l will be replaced by the average of the computed values at l+1 and l-1, and this will be done this for all j. This procedure is called as “lightening.”

Another nonuniform application of the control terms is referred to as “tightening.” Herein the control is applied only along the indicated line or lines, and is deactivated elsewhere. Tightening should not be used on the same face as lightening. Tightening is a very powerful command, and should be used with caution.

If neither lightening or tightening is desired, “no” should be entered in columns 69-70. If either lightening or tightening is desired, “ye” should be placed in those columns.

Control should never be activated on a face which has coincident points. Where points are coincident, certain derivatives are undefined. The calculation of the RHS terms requires all derivatives up through second order. Division by zero will result.

Elliptic grid generation is a complicated process, which sometimes works and sometimes doesn't. The user should have some general guidelines to predict whether a case will work or not. These guidelines will allow the generation of a grid in a series of runs. The user should first seek to simply make the grid

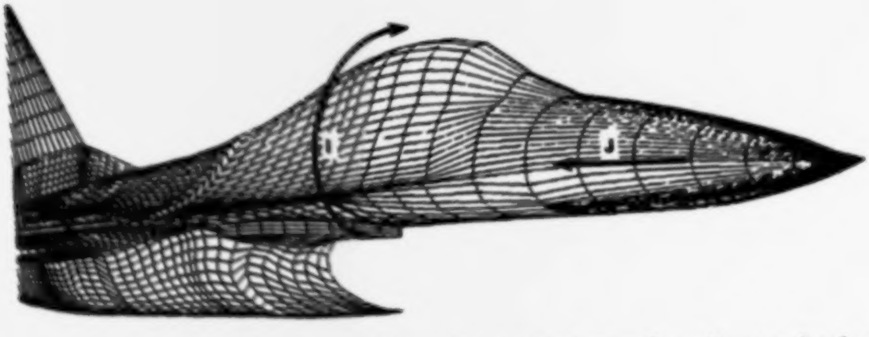


Figure 8.– Surface grid on airplane fuselage with strake. Index j goes from front to back and l goes around from bottom to top in this example.

generator work on a grid which bears some resemblance to the grid that is wanted. The desired grid can then be approached by gradually adjusting the input parameters.

Two such guidelines can be given. First, for each face having control terms activated, the user should calculate the physical distance from a typical point on that face to its correspondent point the opposite face. That distance should then be divided by the number of intervals on the line connecting those points, yielding what would be the spacing on that line if that spacing was uniform. The user should compare that uniform spacing to the spacing being requested in columns 27-38 on the "face" line. For the first try at making the grid generator work, the requested spacing should be between one-half and one-tenth of the uniform spacing. That should work. Once that first value has worked, the user who desires much smaller spacing at the wall can then reduce the requested spacing in increments. Just how much it can be reduced is a very problem-dependent matter, and is impossible to predict generally. The symptom of not working, of course, is that the iterative grid-generation process will not converge.

The second guideline requires calculating the aspect ratio of the grid cells at the face. The user should divide the greatest dimension of a cell on the given surface grid by the height being requested. It is recommended that that aspect ratio not be less than one. That is, cells on the wall should not be taller than they are wide. For the first try, as in the preceding paragraph, that ratio should be no larger than about 10. Once that has worked the user may increase that ratio in increments, by reducing the normal distance given in columns 27-38. Grids have been generated on 64-bit machines (e.g., CRAY X/MP and CRAY-2) with aspect ratios as large as 10,000:1.

13. The "norm/sect" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
1	1	1-10	k	"norm/sect="
1	2	11-13	i	value of the index locating first point
1	3	14	k	","
1	4	15-26	f	cell height at first point
1	5	27	k	","
1	6	27-30	i	value of the index locating second point
1	7	31	k	","
1	8	32-43	f	cell height at second point
1	9	44	k	","

1	10	45-47	i	value of the index locating third point
1	11	48	k	“-”
1	12	49-60	f	cell height at third point
2	1	1-10	k	“norm/sect=”
2	2	11-13	i	value of the index locating fourth point
2	3	4	k	“-”
2	4	15-26	f	cell height at fourth point
2	5	27	k	“-”
2	6	28-30	i	value of the index locating fifth point
2	7	31	k	“-”
2	8	32-43	f	cell height at fifth point
2	9	44	k	“-”
2	10	45-47	i	value of the index locating sixth point
2	11	48	k	“-”
2	12	49-60	f	cell height at sixth point
3	1	1-10	k	“norm/sect=”
3	2	11-13	i	value of the index locating seventh point
3	3	14	k	“-”
3	4	15-26	f	cell height at seventh point
3	5	27	k	“-”
3	6	28-30	i	value of the index locating eighth point
3	7	31	k	“-”
3	8	32-43	f	cell height at eighth point

3	9	44	k	“-”
3	10	45-47	i	value of the index locating ninth point
3	7	48	k	“-”
3	8	49-60	f	cell height at ninth point

```
norm/sect=001-    3.       -020-  37.3
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

Whether there are other lines of input describing the face is dependent upon what values appear on the “face” line. If a form other than “n-i-stations” is chosen for columns 27-38, the input line(s) shown in the table above are not needed and should not appear. If the form “n-i-stations” is used, a line or lines as shown above should immediately follow the “face” line, giving the values for cell height which make up the piecewise linear function. Only the lines needed must appear. That is, if the number of stations at which the linear function is defined is less than seven, then the third line should be deleted. If the number of stations at which the linear function is defined is less than four, then both the second and the third lines should be deleted.

14. The “lighten/tighten” line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-2	c/i	“no” or the # of first sub. lightenings
	2	3	k	“-”
	3	4	c	first subscript: “j” or “k” or “l”
	4	5-13	k	“-lighten-”
	5	14-15	c/i	“no” or the # of second sub. lightenings
	6	16	k	“-”
	7	17	c	second subscript: “j” or “k” or “l”
	8	18-26	k	“-lighten-”

9	27-28	c/i	“no” or the # of first sub. tightenings
10	29	k	“-”
11	30	c	first subscript: “j” or “k” or “l”
12	31-39	k	“-tighten-”
13	40-41	c/i	“no” or the # of second sub. tightenings
14	42	k	“-”
15	43	c	second subscript: “j” or “k” or “l”
16	44-51	k	“-tighten-”

```
no-j-lighten-03-l-lighten-no-j-tighten-no-l-tighten
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

With this line, as well as with the the preceding “norm/sect” line, its presence or absence depends upon what is given on the “face” line. If columns 69-70 on the face line contain “no,” there will be no “lighten/tighten” line. If, however, columns 69-70 of the face line contain “ye,” then a “lighten/tighten” line must appear.

One could be forgiven for being confused by this input line. The information it is trying to convey is complicated. Columns 1-2 tell whether there is a value or values of the first index at which averaging of the RHS terms in the direction of the first index is to take place, for all values of the second index. Conversely, columns 14-15 tell whether there is a value or values of the second index at which averaging of the RHS terms in the direction of the second index is to take place, for all values of the first index. The example above refers to a face number three, where j and l are the running indices. It says that there is no value of j at which the RHS terms are to be averaged in the j direction, for all values of l. But it says that there are three values of l at which the RHS terms are to be averaged in the l direction, for all values of j.

Locations of the lines along which tightening is to be applied are given similarly. The data in columns 27-28 determine whether there are any values of the first index along which tightening is to be applied for all values of the second index. The data in columns 40-41 determine whether there are any values of the second index along which tightening is to be applied for all values of the first index.

15. The “lighten-at” line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-11	k	“lighten-at-”
	2	12	c	name index where lighten: “j” or “k” or “l”
	3	13	k	“=”
	4	14-16	i	value of index where lighten

and continuing across as per the example below:

```
lighten-at-j=001-002-003-004-005-006-007-008-009-010
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

16. The “tighten-at” line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-11	k	“lighten-at-”
	2	12	c	name index where tighten: “j” or “k” or “l”
	3	13	k	“=”
	4	14-16	i	value of index where tighten

and continuing across as per the example below:

```
tighten-at-j=001-002-003-004-005-006-007-008-009-010
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

These two lines should appear only as called for in the “lighten/tighten” line.

Summarizing now, there is an outside loop on the block number, and within that there is an intermediate loop on the face number. For each face, in numerical order, there must be a “face” line. Then, depending on whether they are called for in the "face" line, there will be "norm/sect" line(s) and a "lighten/tighten" line. Then if called for in the "lighten/tighten" line, there will be a "lighten-at" or "tighten-at" line.

At this point all of the data pertaining to the entire face have been read. It is time to go into the innermost loop on section number. For purposes of determining the x,y,z locations of the points on the boundary faces of the block, those faces can be divided into as many as ten sections. Chapter III listed the seven different treatments 3DGRAPE offers for locating boundary points, and any of those treatments may be applied to each section. But the user should keep in mind that in most problems most of the faces will consist of only one section. Thus in what follows "section" can usually be read as "face."

The preceding input lines were given ordinal numbers in the section headings (e.g., 16. The "tighten-at" line). There is some correspondence between those numbers and the placement of their respective input lines, with discrepancies due to repeated or deleted lines. But here ends any semblance of such order, since the boundary treatments listed below may be applied to any section of any face.

The following specifications for boundary treatment of sections of faces all include some indication of the range of indices to which those treatments apply. It is the user's responsibility to check those ranges to make sure that they add up to treatment of the entire face. It would be quite possible to divide a face into sections by index limits and leave holes untreated or have overlapping treatments. Overlapping treatments are inelegant, but rarely cause problems. Leaving holes untreated, however, should be avoided.

A collateral problem is treating the edges of the block, each of which is the intersection of two faces. Here again they might be treated twice, as part of two different faces, or they might be not treated at all. Redundant treatment is clumsy, but is rarely wrong. In such cases the treatment associated with the face having the highest face number will take precedence. But failing to treat an edge in any way will be a sure cause of failure.

Immediately following the input(s) pertaining to the face, there should follow one of the following boundary treatment inputs for each section on the face, with no intervening blank lines.

17. The "read-in-fixed" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-18	k	"read-in-fixed-xyz-"
	2	19	c	first index on the face: "j" or "k"
	3	20-25	k	"-from-"
	4	26-28	i	starting value of first index

	5	29-32	k	"-to-"
	6	33-35	i	ending value of first index
	7	36	k	"-"
	8	37	c	second index on the face: "k" or "l"
	9	38-43	k	"-from-"
	10	44-46	i	starting value of second index
	11	47-50	k	"-to-"
	12	51-53	i	ending value of second index

```
read-in-fixed-xyz-j-from-001-to-025-k-from-001-to-025
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

This treatment is used for inputting a fixed boundary surface, typically the shape or part of the shape about which or inside of which the user desires to make a grid. As stated previously, these points on this surface must be distributed properly by some other device prior to input here. The points on this surface must be distributed with two running indices, as is typical of any surface mapping into the side of a computational cube. Those x,y,z data are not actually read from this file, file10. Instead, upon reading the "read-in-fixed" input line, 3DGRAPE looks to file11 from which it actually reads the data. File11 is described in the following section. After reading x,y,z data for this section of this face from file11, 3DGRAPE returns to file10 and continues reading.

18. The "plane-normal-to" lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-16	k	"plane-normal-to-"
	2	17	c	axis to which perpendicular: "x" or "y" or "z"
	3	18-26	k	"-axis-at-"
	4	27	c	axis to which perpendicular: "x" or "y" or "z"
	5	28	k	"="

6	29-40	f	location on axis
7	41	k	"-"
8	42	c	first index on the face: "j" or "k"
9	43-48	k	"-from-"
10	49-51	i	starting value of first index
11	52-55	k	"-to-"
12	56-58	i	ending value of first index
13	59	k	"-"
14	60	c	second index on the face: "k" or "l"
15	61-66	k	"-from-"
16	67-69	i	starting value of second index
17	70	k	"-"
1	1-6	k	"...to-"
2	7-9	i	ending value of second index

```
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The points in this section, as defined by the given indices, will be constrained to lie on a plane normal to the indicated axis, at the indicated point on that axis. The distribution of points on that plane will be found by extrapolating from the elliptic grid in the interior of the block. This is done in a manner such that grid lines coming from the interior of the block and intersecting the plane do so at right angles.

The distribution of points on that plane given in the initial conditions, however, will be nonsense. The user plotting initial conditions should ignore points on planes such as this.

19. The "cylinder-about" lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
1	1	1-15	k	"cylinder-about-"
1	2	16	c	name of axis: "x" or "y" or "z"
1	3	17-27	k	"-axis-from-"
1	4	28	c	name of axis: "x" or "y" or "z"
1	5	29	k	"="
1	6	30-41	f	starting value on axis
1	7	42-45	k	"-to-"
1	8	46	c	name of axis: "x" or "y" or "z"
1	9	47	k	"="
1	10	48-59	f	ending value on axis
1	11	60	k	"="
1	12	61	c	name of index along cylinder: "j" or "k" or "l"
1	13	62-68	k	"-along-"
2	1	1-13	k	"-axis-from-"
2	2	14-16	i	starting value of index along
2	3	17-20	k	"-to-"
2	4	21-23	i	ending value of index along
2	5	24	k	"="
2	6	25	c	name of index along cylinder: "j" or "k" or "l"
2	7	26-38	k	"-around-from-"

2	8	39-41	i	starting value of index along
2	9	42-45	k	"-to-"
2	10	46-48	i	starting value of index around
2	11	49-60	k	"-with-angle="
3	1	1-3	k	"..."
3	2	4-15	f	starting value of angle around (in degrees)
3	3	16-25	k	"-to-angle="
3	4	26-37	f	ending value of angle around (in degrees)
3	5	38-45	k	"-radius="
3	6	46-57	i	radius of cylinder

```
cylinder-about-x-axis-from-x=100.        -to-x=     750.    -j-along-
...axis-from-002-to-033-l-around-from-002-to-021-with-angle=
   -90.        -to-angle= +90.       -radius=     500.
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The points in this section, as defined by the given indices, will be constrained to lie on the surface of a cylinder. That cylinder must have its axis coincident with one of the coordinate axes. 3DGRAPE needs to know the limits of the cylinder in the axial direction. Note that the "starting value" on the axis should correspond to the starting value of the index running along the axis, and the "ending value" on the axis should correspond to the ending value of that index. The index limits should be given in increasing fashion, i.e., the ending limit of the index should be greater than the starting value. But the physical problem may demand that the values on the axis corresponding to those indices be given in decreasing fashion, i.e., the ending value on the axis may be smaller than its starting value. That is acceptable.

The extrapolation to the cylinder is done in the following manner. For each point on the cylinder, two locations are found. The first is found by striking a ray from the axis through the point nearest the cylinder and onto the cylinder. The second location is found by linearly extrapolating from the two points nearest the cylinder and onto the cylinder. Those two locations are then averaged.

The cylinder need not displace the entire 360°. For example, in an aerodynamic application which assumes no yaw, the grid typically covers only one side, requiring a cylindrical section of 180°. Thus starting and ending values of the angle around the cylinder are input. Those angles are defined according to the increasing index convention for right-handed coordinate systems, and a decreasing index convention for left-handed systems. An alternate explaination of that angle definition is as follows. The cylinder's axis is one of the coordinate axes. The user should imagine his eye far out on the positive end of that axis, looking back toward the origin at the entire grid. The user will then be looking at a coordinate plane in which lie the two other axes. That plane should be rotated, and the entire grid with it, about the cylindrical axis until the positive end of one of those other two axes points to the right and the other positive end points up. The user can then imagine a conventional 2-D polar coordinate system on that plane, with the angle equal to zero on the right and increasing in counterclockwise fashion. It is with respect to that angle that the starting and ending angles entered in columns 4-15 and 26-37 of the third input line are measured.

The axis values and the angles are used only for locating the initial conditions. Thus great precision is not required.

Note that the starting and ending values of the index running around the axis should be given in increasing order, i.e., the ending value must be greater than the starting value. But the starting and ending values of the angle need not be so ordered; the physical problem may require that they be ordered backwards. That is acceptable.

20. The "ellipsoid" line

Line no.:	Field no.:	Column nos:	Datum type:	Description:
1	1	1-17	k	"ellipsoid-x-cent="
1	2	18-29	f	x-coordinate of center of ellipsoid
1	3	30-37	k	"-y-cent="
1	4	38-49	f	y-coordinate of center of ellipsoid
1	5	50-57	k	"-z-cent="
1	6	58-69	f	z-coordinate of center of ellipsoid
2	1	1-10	k	"...x-semi="
2	2	11-22	f	length of semi-span in x-direction
2	3	23-30	k	"-y-semi="
2	4	31-42	f	length of semi-span in y-direction
2	5	43-50	k	"-z-semi="
2	6	51-62	f	length of semi-span in z-direction

2	7	63	k	"-"
2	8	64	c	name of first index: "j" or "k"
2	9	65-70	k	"-from-"
3	1	1-3	k	"..."
3	2	4-6	i	starting value of first index
3	3	7-10	k	"-to-"
3	4	11-13	i	ending value of first index
3	5	14	k	"-"
3	6	15	c	name of second index: "k" or "l"
3	7	16-21	k	"-from-"
3	8	22-24	i	starting value of second index
3	9	25-28	k	"-to-"
3	10	29-31	i	ending value of the second index

```
ellipsoid-x-cent=100.        -y-cent=   0.       -z-cent=   0.
...x-semi=  500.      -y-semi=   500.     -z-semi=  500.      -j-from-
...002-to-018-l-from-002-to-021
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The points on a face, or on a section of a face, may be constrained to lie on the surface of an ellipsoid. A sphere, of course, is a special case of an ellipsoid. The center of the ellipsoid may lie anywhere, and that location is given on the first line. The ellipsoid must, however, have its semi-axes parallel with the coordinate axes. The shape of the ellipsoid is defined by the length of the semi-axes. The length of the semi-axis in the x-direction, i.e., the distance from the center to the surface measured in the x-direction, is given in columns 11-22 of the second line. The other semi-axes are given similarly.

A problem arises when the initial conditions for points on the ellipsoid are too far from what should be their location in the converged solution. This problem can cause the grid solution to diverge. The solution to this problem is to require, because of programming constraints, that (1) only even-numbered faces be ellipsoids, and (2) the opposing odd-numbered face be of the "read-in-fixed" type. Given this situation, the initial conditions on the ellipsoid are found by striking a line from the center of the ellipsoid through each point on the "read-in-fixed" face, and locating the corresponding point on the ellipsoid at the place where that line pierces it.

The extrapolation of points onto the ellipsoid, during the grid-generation solution process, is done by a simple linear extrapolation from the two points nearest the ellipsoid.

21. The "collapsed-to-an-axis" lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
1	1	1-13	k	"collapsed-to-"
1	2	14	c	name of the axis: "x" or "y" or "z"
1	3	15-25	k	"-axis-from-"
1	4	26	c	name of the axis: "x" or "y" or "z"
1	5	27	k	"="
1	6	28-39	f	starting value on the axis
1	7	40-43	k	"-to-"
1	8	44	c	name of the axis: "x" or "y" or "z"
1	9	45	k	"="
1	10	46-57	f	ending value on the axis
1	11	58	k	"-"
1	12	59	c	name of index along axis: "j" or "k" or "l"
1	13	60-66	k	"-along-"
2	1	1-13	k	"...axis-from-"
2	2	14-16	i	starting value of the index along axis
2	3	17-20	k	"-to-"
2	4	21-23	i	ending value of the index along axis

2	5	24	k	“-”
2	6	25	c	name of index around axis: “j” or “k” or “l”
2	7	26-38	k	“-around-from-”
2	8	39-41	i	starting value of index around axis
2	9	42-45	k	“-to-”
2	10	46-48	i	ending value of index around axis

```
collapsed-to-x-axis-from-x=    0.      -to-x= -400.      -k-along-
...axis-from-002-to-031-l-around-from-001-to-022
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

Certain topologies, such as spherical or cylindrical grids, give rise to the need for a face, or a section of a face, to be collapsed to an axis. This input option allows that treatment. Note that the points on the axis are found by extrapolating to the axis, and so the distribution of points on the axis is that which results from the elliptic solution. Elliptic grids tend to be uniformly distributed, absent the effect of control terms. Thus the distribution of points on faces collapsed to axes tends to be uniform.

As faces collapsed to axes have many coincident points, control terms should not be activated thereon.

The axis values given here are used only for locating the initial conditions. Thus great precision is not required.

The starting and ending values of the indices should be given in increasing order, i.e., the ending values should be larger than the starting values. This sometimes means that the corresponding starting and ending values on the axis must be given in decreasing order, i.e., with the ending values less then the starting values. That is acceptable.

22. The “collapsed-to-a-point” lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
1	1	1-21	k	“collapsed-to-point-x=”
1	2	22-33	f	x-coordinate of the point

1	3	34-36	k	"-y="
1	4	37-48	f	y-coordinate of the point
1	5	49-51	k	"-z="
1	6	52-63	f	z-coordinate of the point
1	7	64-69	k	"-with-"
2	1	1-3	k	"..."
2	2	4	c	name of first index: "j" or "k"
2	3	5-10	k	"-from-"
2	4	11-13	i	starting value of first index
2	5	14-17	k	"-to-"
2	6	18-20	i	ending value of first index
2	7	21	k	"-"
2	8	22	c	name of second index: "k" or "l"
2	9	23-28	k	"-from-"
2	10	28-31	i	starting value of second index
2	11	32-35	k	"-to-"
2	12	36-38	i	ending value of second index

```
collapsed-to-point-x=  750.      -y=   0.       -z= 0.         -with-
...j-from-001-to-001-l-from-001-to-022
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

Because all points on this section are coincident, control must not be activated here.

23. The “match-to-face” lines

Line no.:	Field no.:	Column nos:	Datum type:	Description:
1	1	1-14	k	“match-to-face-”
1	2	15	i	face number of other face
1	3	16-22	k	“-block-”
1	4	23-24	i	block number of other face
1	5	25-30	k	“-this-”
1	6	31	c	name of first index on this face: “j” or “k”
1	7	32-37	k	“-from-”
1	8	38-40	i	starting value of first index on this face
1	9	41-44	k	“-to-”
1	10	45-47	i	ending value of first index on this face
1	11	48-53	k	“-this-”
1	12	54	c	name of second index on this face: “k” or “l”
1	13	55-60	k	“-from-”
1	14	61-63	i	starting value of second index on this face
1	15	64-67	k	“-to-”
1	16	68-70	i	ending value of second index on this face
2	1	1-9	k	“...-that-”
2	2	10	c	first index on that face: “j” or “k” or “l”
2	3	11-16	k	“-from-”
2	4	17-19	i	starting value of first index on that face

2	5	20-23	k	"-to-"
2	6	24-26	i	ending value of first index on that face
2	7	27-32	k	"-that-"
2	8	33	c	second index on that face: "j" or "k" or "l"
2	9	34-39	k	"-from-"
2	10	40-42	i	starting value of second index on that face
2	11	43-46	k	"-to-"
2	12	47-49	i	ending value of second index on that face

```
match-to-face-1-block-02-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

This boundary treatment allows this section of this face to be matched to (1) any other section of this face, or (2) any section of another face of this block, or (3) any section of any face of any other block. That match will produce a block-to-block type boundary where the surface floats with the solution of the grid-generation equations. Grid line slope and spacing will be continuous across this surface. Note that this surface is double-stored, i.e., it exists in memory identically as part of both coincident faces.

The range of indices defining "this" section must match with the range of indices defining "that" section. Note that while the first index on "this" face must be j or k and its second index must be k or l, any index could be the first index on "that" face and any other index could be its second index. The starting and ending values on "this" face must be given in increasing fashion, i.e., the ending values must be greater than the starting values. But the corresponding indices on "that" face may run in whatever direction is appropriate. Note that the information given here must essentially be given twice—once here in these input lines describing "this" face of "this" block, and also in the input lines describing "that" face of "that" block.

The initial conditions on this section of this face are nonsense, and should not be plotted.

C. FILE11 - BODY DEFINITION ARRAYS

It was stated in the previous section that during its input phase 3DGRAPE reads through file10 until it encounters a "read-in-fixed" input line. At that point it suspends reading from file10 and begins reading the fixed surface from file11. When it is finished reading that fixed surface from file11, it returns to reading from file10. This cycle will be repeated as many times as there are "read-in-fixed" input lines.

Thus file11 must contain x,y,z coordinates of as many fixed surfaces as there are "read-in-fixed" input lines.

For each fixed-surface read from file11, it must contain: (1) a header line introducing the x-coordinates, (2) the x-coordinates, (3) a header line introducing the y-coordinates, (4) the y-coordinates, (5) a header line introducing the z-coordinates, and (6) the z-coordinates. No intervening blank lines are allowed. This cycle of six should be repeated for each fixed surface.

The header lines introducing the coordinates are of the form:

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-9	k	"complete-"
	2	10	c	name of coordinate: "x" or "y" or "z"
	3	11-23	k	"-for-section-"
	4	24-25	i	section number
	5	26-34	k	"-of-face-"
	6	35	i	face number
	7	36-45	k	"-of-block-"
	8	46-47	i	block number

```
complete-x-for-section-01-of-face-3-of-block-01
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The actual x- or y- or z-coordinates begin on the line immediately following their header. They are read in 6F12.0 format (which means that the placement of the decimal point is at the user's discretion). That format is repeated for subsequent lines as many times as needed. The points should be ordered with the first subscript (see the "read-in-fixed" line in file10, described in Section B, Subsection 17) varying most rapidly as the "inner loop," and the second subscript varying most slowly as the "outer loop."

D. FILE16 – CONTROL SCALARS FOR RESTART

3DGRAPE has a restart capability. The user can let the grid generator run a while, examine the resulting grid, change some things, and then run it some more. A restart capability is simple in concept, but difficult to program. It is not trivial to decide what should be reset to its starting value, what should be continued as at the end of the previous run, and what the user should be allowed to change.

If a run of 3DGRAPE is to be restarted, the first requirement is that a restart file be written as file15 by the starting run. This is done by placing a "ye" in columns 19-20 of the "write-for-restart" input line in file10. This restart file, file15, is not the same as the main grid output file, file14. The restart file is much larger, since it must contain not only the x,y,z of the grid, but also other data, including the current values of the RHS terms. For the restart run, this file is rewound and read in as file17.

3DGRAPE restarts by reading the first two input lines, as described in Section A of this Chapter, from either the terminal in the case of an interactive computer system or the main job input stream in the case of a batch system. It then reads a file of input scalars, called file16, which is very similar to file10. It was intended for users to create their file16 by copying, renaming, and modifying file10.

Input parameters can be keep the same by simply not modifying them in file16 after the copy operation. Some variables must always remain unchanged, and to effect that the input lines which might have changed them must be removed. An example of this is the dimension sizes of the blocks. Storage is allocated based upon those sizes; to change them at a restart would result in chaos.

But there are many things which may be modified at a restart. The iteration schedule may be completely rewritten. File names may be changed. Although whether control is activated at a certain face may not be changed, the requested cell height and all other parameters on the "face" input line may be changed. The global switch activating or deactivating all control terms may be used. The number of sections into which a face is divided, and the index limits which determine the sizes of those sections, may be changed. The treatments specified for locating boundary points may be changed. Although this author is confident that the particular operations the restart capability offers are programmed correctly, it is impossible to exhaustively predict the effects of all possible combinations of them in operation. The philosophy here is to "give [the user] enough rope to hang himself." Caveat emptor.

The following table lists the input lines in file16. As compared to file10, one new line has been added and it is described. One line has been modified, and it is described. An example of a file16 can be seen in the chapter on the first example problem.

Input line(s):	Its (their) disposition in file16:
"run-comment"	These two lines are read in, and their contents will overwrite the stored run comment.
"filename-17-input"	This is a new type of input line. See below.
"number-of-parts"	This line is a modification of what was the "number-of-blocks" line in file10. See below. It is not permitted that the number of blocks change in a restart.

"iterations"	This line is (these lines are) read and used just as in file10. It (they) may be changed.
"filename-11"	This line is read and used just as in file10. The file11 name or its contents may be changed.
"filename-14"	This line is read and used just as in file10. It may be changed.
"write-for-restart"	This line is read and used as in file10. But the restart file which this input line will cause to be written will be for a further restart, subsequent to this restart run. Thus the filename should be different than is used for this run's file17.
"relaxation-param"	This line is read and used just as in file10. It may be changed.
"block-comment"	This line is read and used just as in file10. It may be changed.
"dimension"	This line should be removed from file16. No data on it are permitted to change.
"handedness"	This line should be removed from file16. No data on it are permitted to change.
"polar-axis"	This line should be removed from file16. No data on it are permitted to change.
"face"	This line is read and used as in file10. But there must not be any modification of whether the control is activated on this face. See the "normal" parameter. Storage is allocated based upon which faces have active control; chaos would result from adding to or deleting from this collection of faces. However, it is permitted to vary the specified height of cells on faces with active control. It is permitted to change the other parameters on this line.
"norm/sect"	This line is read and used just as in file10. It may be added, deleted, or changed as appropriate.
"lighten/tighten"	This line is read and used just as in file10. It may be added, deleted, or changed as appropriate.
"lighten-at"	This line is read and used just as in file10. It may be added, deleted, or changed as appropriate.
"tighten-at"	This line is read and used just as in file10. It may be added, deleted, or changed as appropriate.

"read-in-fixed"	This line is read and used just as in file10. It may be changed. The boundary points for this face will be reread from file11 as before. Thus the actual points on the boundary face may be changed by modifying file11.
"plane-normal-to"	These lines are read and used just as in file10. They may be changed.
"cylinder-about"	These lines are read and used just as in file10. They may be changed.
"ellipsoid"	These lines are read and used just as in file10. They may be changed.
"collapsed-to-an-axis"	These lines are read and used just as in file10. They may be changed.
"collapsed-to-a-point"	These lines are read and used just as in file10. They may be changed.
"match-to-face"	These lines are read and used just as in file10. They may be changed.

The new "filename-17" line is as described below:

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-18	k	"filename-17-input="
	2	19-33	n	filename for restart file to be read

```
filename-17-input=restartex1
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

The "number-of-parts" line, a modification of what is the "number-of- blocks" line in file10, is as described below:

Line no.:	Field no.:	Column nos:	Datum type:	Description:
	1	1-38	k	"number-of-parts-in-iteration-schedule="
	2	39-40	i	number of parts in the iteration schedule

```
number-of-parts-in-iteration-schedule=02
1234567890123456789012345678901234567890123456789012345678901234567890
0000000001111111111222222222233333333334444444444555555555566666666667
```

V. RUNNING 3DGRAPE AND EVALUATING ITS OUTPUT

A. A "GAME PLAN" FOR RUNNING 3DGRAPE

There is in all users, it seems, a tendency to want to leap immediately to the finished grid. But wise users will save time in the long run by pursuing their goals in steps. These steps constitute a game plan for running 3DGRAPE, and its use is most strongly recommended for all users at all times. These steps are

1. Set the desired number of iterations to zero. See the "iterations" input line in file10. This will cause 3DGRAPE to simply write out the initial conditions in file14 as though it were a finished grid. This initial condition grid should be examined using 3-D graphics to ascertain that the user's read-in-fixed boundary data have been entered correctly. All faces having points constrained to lie on cylinders, ellipsoids, and lines, and collapsed to a point should be similarly checked. Some representative interior surfaces should be plotted to see whether the initial conditions inside the blocks are being done right.

   Where faces have their points constrained to lie on planes, those points should lie on those planes. But there is no guarantee that the distribution of those points on those planes will be reasonable in the initial conditions. The distribution of points on those faces will probably be nonsense. Thus examination of such faces in the initial condition grid should be done with caution. For similar reasons, little credence should be given to "match-to-face" type faces in the initial condition grid.

   Sample input for this zero iteration run:

   ```
   number-of-blocks=03-number-of-parts-in-iteration-schedule=01
   iterations=000-control=no-coarse/fine=coarse
   ```

   In this sample input, the coarse/fine parameter is ignored since no iterations are performed.

2. Next, the user should obtain a converged Laplacian solution. This will validate all of the the boundary treatments, a very desirable thing to have done when later debugging the case with control activated. A Laplacian solution may be obtained by simply turning the global switch off in columns 24-25 of the "iterations" input line(s). Sample input for this Laplacian solution:

   ```
   number-of-blocks=03-number-of-parts-in-iteration-schedule=02
   iterations=100-control=no-coarse/fine=coarse
   iterations=020-control=no-coarse/fine=fine
   ```

   Note that this sample input will exercise the coarse/fine procedure. The Laplacian grid should be thoroughly examined using interactive graphics.

3. Finally, the user should obtain the controlled solution. The following is sample input for that purpose:

   ```
   number-of-blocks=03-number-of-parts-in-iteration-schedule=03
   ```

```
iterations=020-control=no-coarse/fine=coarse
iterations=150-control=ye-coarse/fine=coarse
iterations=075-control=ye-coarse/fine=fine.
```

The first part in this sample, 20 iterations with no control, is for the purpose of letting the initial conditions, which are pretty strange for some boundary treatments, become reasonable before the control terms are activated. If the control terms are suddenly activated on a grid which is not yet somewhat reasonable, instability can result.

In the sample input, above, control terms are turned on for 150 more coarse steps in the second part, which will hopefully yield a converged controlled solution over the coarse grid. The third iteration step causes that coarse solution to be interpolated and smoothed.

The above sample input is not binding. The user may tailor the iteration count, the use or nonuse of the coarse/fine procedure, the use or nonuse of the restart capability, etc., to any particular problem.

For difficult cases the final controlled solution might even be obtained in a series of steps. See the discussion of guidelines for that purpose in the final three paragraphs of subsection 12 in Chapter IV, the discussion of the "face" input line. Eugene Tu (private communication), Ames, reports that smaller values of cell height and smaller values of the abc parameter can be achieved by a series of restarts, than by starting initially with the desired values.

B. READING THE PRINTOUT

This author would have liked to include in this manual a sample printout, but its length made that impossible. The printout for Example 1, the simulated helicopter fuselage case, would have required 26 pages in this form. But an attempt will be made here to describe that printout. (In the case of an interactive computer system, these are the data which will appear on standard output, whether they come up on a screen or are redirected to a file.)

First on the printout are the prompts for the first two lines of input, as described in Chapter IV, along with the user's responses. Next is a banner which gives the name of the program (3DGRAPE), this author's name and address, and the run-comment information from the first two lines of input in file10 (or file16 in the case of a restart). Following that is a listing of the remainder of file10 (or file16), double spaced, with line numbers. Notation is made when data are read from file11.

Next in the printout is a simple trace of the iteration count, consisting of one line per iteration. If the program blows up from lack of adequate CPU time, this trace can be easily used to calculate how much time was required.

Next are convergence histories for each of the blocks, in turn. The header introducing each history gives the information from the appropriate block-comment input line. Each history lists five pieces of data for each iteration:

1. The iteration count—this should simply increase to the required number, which is the sum of the iterations specified on the "iterations" input lines. These numbers are cumulative over all new starts and restarts. A truncation of this series indicates that the program sensed that the solution was blowing up.

2. Maxmove—the greatest distance moved by any point in this block during this iteration. This datum might rise at the start of each part in the iteration schedule, but it should eventually decay toward zero.

3. Avemove—the average of the distances moved by all moving points in this block during this iteration. This datum also might rise at the start of each part in the iteration schedule, but it also should eventually decay toward zero. Avemove should be significantly (typically two or more orders of magnitude) less than Maxmove.

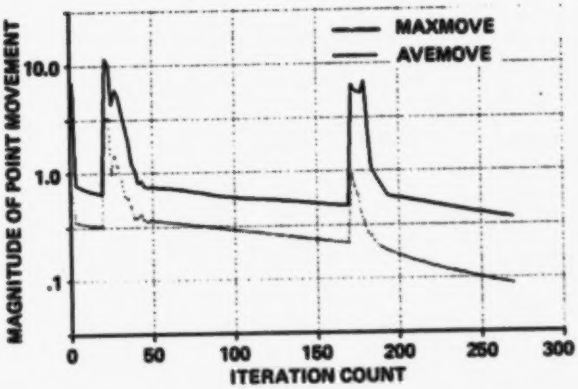
4. Pqrmax—the maximum absolute value of any RHS term. This datum should not decay to zero. Instead, it should increase and then level off to a constant value.

5. Pqrcor—the maximum absolute value of the correction on any RHS term. This datum should decay toward zero, and be significantly less than Pqrmax.

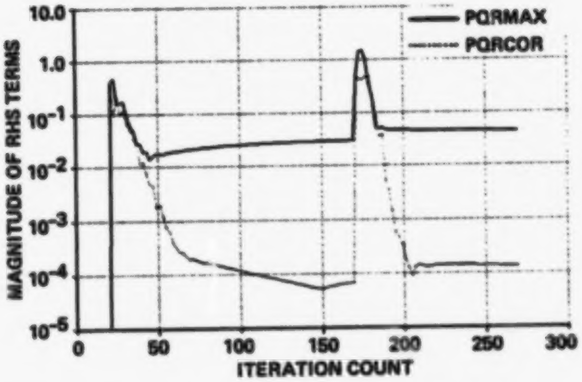
The printout concludes with a notation of the output and restart files which have been written.

Figure 9 shows plots of the convergence history for block 2 of Example 1, the simulated helicopter fuselage. This case was run using the file10 input data shown in Appendix A, file11 input data generated by the program shown in Appendix B, and file16 (restart) input data shown in Appendix C. Note from the input data that breaks between parts in the iteration schedule occur at 20 iterations and at 170 iterations, and that the restart run begins after 245 iterations. All plotted functions jump upward at 20 and 170 iterations. A jump occurs at 20 iterations because it is there that the RHS terms are "turned on," and at 170 iterations because there the coarse solution is interpolated and iteration begins on the fine grid. At 245 iterations, the start of the restart run, no jump occurs. This indicates that the restart is done correctly. Note in figure 9a that Avemove is everywhere less than Maxmove, and that they both tend toward zero at the right end. For the average movement of all points in that block to be reduced to less than 0.1 unit is commendable in view of the fact that that zone has a cylindrical outer boundary 1000 units in diameter. Note in figure 9b that for the first 20 iterations there are no values given for Pqrmax or Pqrcor, since for those iterations the global switch of control terms is turned off. The Pqrmax curve flattens out to a constant but nonzero value toward the right end. Pqrcor is reduced to a value two and one-half orders of magnitude smaller than Pqrmax, indicating that the program has found the RHS terms which will yield the desired grid behavior at the boundary. This figure shows typical behavior of these functions in a successful run of 3DGRAPE.

3DGRAPE has no facility for creating such a plot. But this author found it a straightforward matter to write a program to literally read the numbers off the printout file and plot them. Creation and use of such a utility is highly recommended.



(a) Maxmove and Avemove, the magnitude of point movement.



(b) Pqrmax and Pqrcor, the magnitude of RHS terms.

Figure 9.– Convergence history for block 2 of Example 1.

C. WHAT TO DO IF THE PROGRAM "BLOWS UP"

A program is said to blow up if it fails to run to completion—if it fails in some catastrophic way.

The first way in which 3DGRAPE might blow up is while it reads the input. If a bad input line is encountered in file10 (or file16 in the case of a restart) or in file11, it will print that line, then in most cases it will print an error message, and then it will quit. Debugging a lengthy input might require fixing many small bugs, and thus require many short runs. For that reason, it was found very convenient to port the program to the IRIS workstation and run it there. Once the input is correct, it and the program are shipped to a more powerful computer for "number-crunching." To correct an error in an input line, refer to the description of that input line in Chapter IV.

The next reason 3DGRAPE might blow up is because it runs out of CPU time. The amount of computer time required is dependent upon many factors, including which computer is being used, the number of mesh points, whether spherical topology is used, whether the coarse-fine procedure is used, how strong the required clustering is, etc. It is very difficult to predict how much CPU time a grid will require, but one example can be given. Example 1, the simulated helicopter fuselage case, calculated on the basis of each coarse iteration counting as 1/27th of a fine iteration, required 0.00002 sec of CPU time per fine iteration per point on a CRAY X/MP.

As stated above, the printout gives a simple trace of the iteration count, printing one line at the conclusion of each iteration. Using this trace, the user whose run ran out of time can easily calculate how much more time would be needed.

3DGRAPE could blow up during the Laplacian solution (see the game plan, above). The symptom of that failure will be Maxmove and Avemove increasing without bound. In this case the input is probably syntactically correct but conceptually wrong. By looking at a zero-iteration solution, or at a Laplacian solution with the iteration count reduced so that it quits before it blows up, the user will probably see that some boundary treatment is wrong. It is very rare for 3DGRAPE to be unable to generate a Laplacian grid with correct boundary treatments.

Last, 3DGRAPE can blow up while generating the controlled grid solution. If it blows up on the first step which has control turned on, it is probably in the calculation of the RHS terms. Such a failure is usually the result of having two read-in-fixed boundary points being coincident. Such a situation requires division by zero.

If 3DGRAPE begins iterating with control turned on, and then Maxmove and Avemove increase without bound and cause it to blow up, the user should refer to the discussion of guidelines for difficult cases in the final three paragraphs of subsection 12 in Chapter IV.

Sometimes 3DGRAPE will run to completion and generate a grid, but graphical examination will reveal that the grid is obviously not suitable. The user should check the the discussion of guidelines for difficult cases discussed in subsection 12 of Chapter IV, and the "handedness" should also be checked. The user who can't be sure of the handedness can simply reverse it and try again.

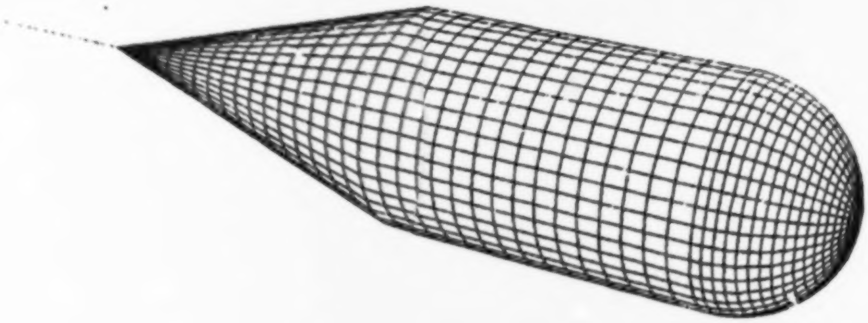
VI. EXAMPLE 1: GRID ABOUT A SIMULATED HELICOPTER FUSELAGE

Example 1 is referred to as a grid about a helicopter fuselage, but in reality the body is a highly simplified analytic shape which bears some vague resemblance to a helicopter fuselage. It does, however, share the same topology as that which might be used for a helicopter fuselage. Thus the input to 3DGRAPE for this case is very similar to what might be used for an actual helicopter fuselage. This simplified shape has the advantage of requiring a small and simple program to generate it, a program which can and does appear in an appendix. This is in opposition to a real helicopter shape which would require sophisticated surface-fitting software, possibly run in an interactive mode, etc. This case has the added advantage of exercising most of the options in 3DGRAPE, making it a valuable example.

The surface grid about this body is shown in Figure 10a. It consists of a hemisphere followed by a cylinder followed by a cone followed by an axis. As in many aerodynamic applications bilateral symmetry is assumed, and thus only one side (the right side) is gridded. These body points are generated by simple analytic means, implemented in the FORTRAN program listed in Appendix B. They are written out in file11 form.

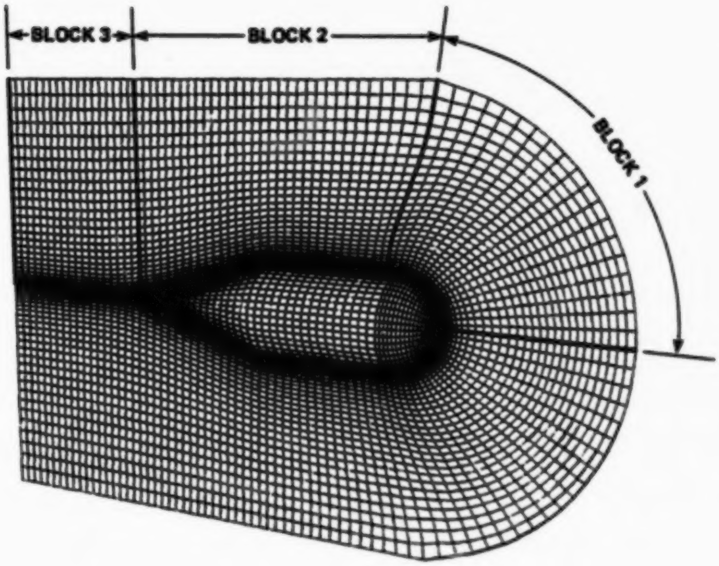
Those same body points are also written out in file14 form, the form of the finished grid, with one point in the radial direction. The data in this form can be viewed using the same graphics software (not supplied as part of 3DGRAPE) as is used to view the finished grid. Thus the body can be checked visually.

Appendix A lists the file10 input data for this case. It can be seen from this input data, and from Figure 10b, that the grid is generated using three blocks—one for the hemispherical nose cap, one about the cylinder and cone, and one about the axis behind. The region near the nose of the fuselage had to be done with the spherical coordinate option because of the presence of a spherical axis. See the "handedness" line for block 1. But it is not recommended that an entire grid be done needlessly with that option; the spherical coordinate option should only be used near the axis. Thus a block boundary was placed at the aft end of the nose cap, at the start of the cylinder.



(a) Fuselage.

Figure 10.– Example 1: grid about analytic shape resembling helicopter fuselage.



(b) Block structure.

Figure 10.– Continued

The second block boundary at the aft end of the cone, at the start of the axis behind, was made necessary by the fact that control was desired on the surface of the fuselage back to the end of the cone, but control was neither desired nor possible on the axis. Since control must be exercised or not on a face-by-face basis, those two regions had to be two different faces. Hence the block boundary and a total of three blocks.

It is this file10 from which most of the examples of input lines in Chapter IV were taken. The iteration schedule, in particular, was discussed there. For all blocks the "initcond" parameter on the "handedness" line is set to "k," since that is the index which proceeds from the body to the outer boundary. No other choice for that parameter would make any sense. Note on the "polar-axis" line for block 1 that the approximate center is given as 100. Since in this case the nose of the body was chosen to be at the origin of the coordinate system, and the hemisphere has a radius of 100 units, the actual geometric center of the hemisphere is at x=100.

In Chapter III, section B, it was stated that the user has the freedom to choose how the indicies run, but once that choice is made the numbering of the faces is set by 3DGRAPE. In every block of this example, the first index, j, has been chosen to run back along the body; that means that in every block face 1 is at the upstream end (the axis for block 1), and face 2 is at the downstream end. The second index, k, has been chosen to run from the body to the outer boundary, so face 3 is the body and face 4 is the outer boundary for all blocks. This leaves the third index, l, to go around, here from lower symmetry plane to upper symmetry plane. Thus face 5 is the lower symmetry plane and face 6 is the upper symmetry plane.

All of the dimension sizes used—19, 31, and 22 in blocks 1 and 3, and 34, 31, and 22 in block 2—are of the form 3n+1 for n an integer greater than or equal to 4. Thus the coarse/fine speedup procedure can be used.

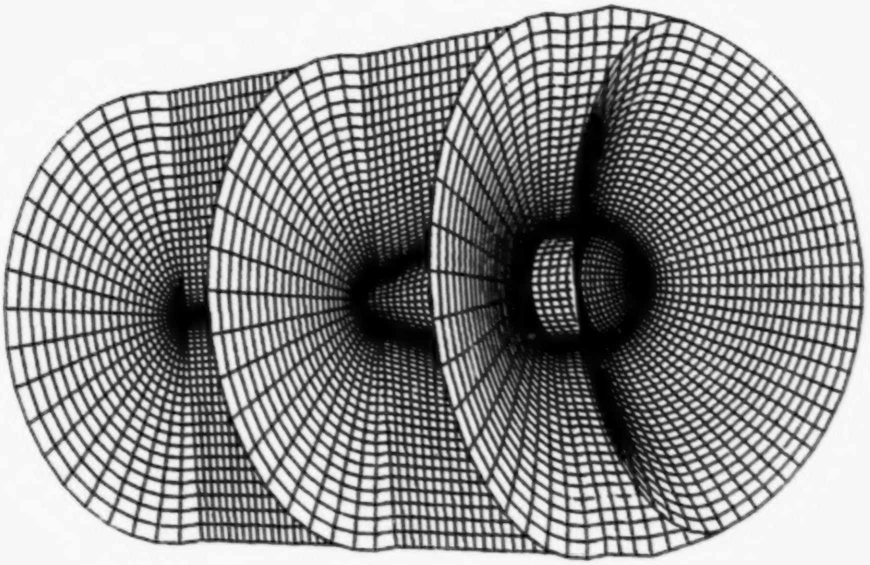
The "face" input lines show that only on the body, i.e., only on face 3 of blocks 1 and 2, is control activated. It is activated there by setting the "normal" parameter to 2.0. In the discussion of the "face" input line (Subsection 12, Section B, Chapter IV) two guidelines were given to help the users predict whether the case will work. The first was a comparison of the "normal" parameter to the uniform spacing along lines normal to boundaries. In this case the body has a radius of 50 units, the outer boundary has a radius of 500 units, giving a nominal distance between of 450 units. The dimension size on k is 31, giving 30 intervals, and a uniform spacing of 15 units. The guideline applied indicates that the "normal" parameter should lie between 1.5 and 7. It does, and the case works.

The second guideline called for a computation of the requested aspect ratio of the grid cells on the body. In this case the longest cells on the body are found on the cylinder part, and are 18.8 units long. A cell height of 2.0 units, as requested by the "normal" parameter, gives an aspect ratio of 9.4. This is within the guideline limits of 1 and 10. The reader should realize, of course, that these guidelines are only for the purpose of helping get started with each case. The "normal" parameter could be reduced greatly if this is desired.

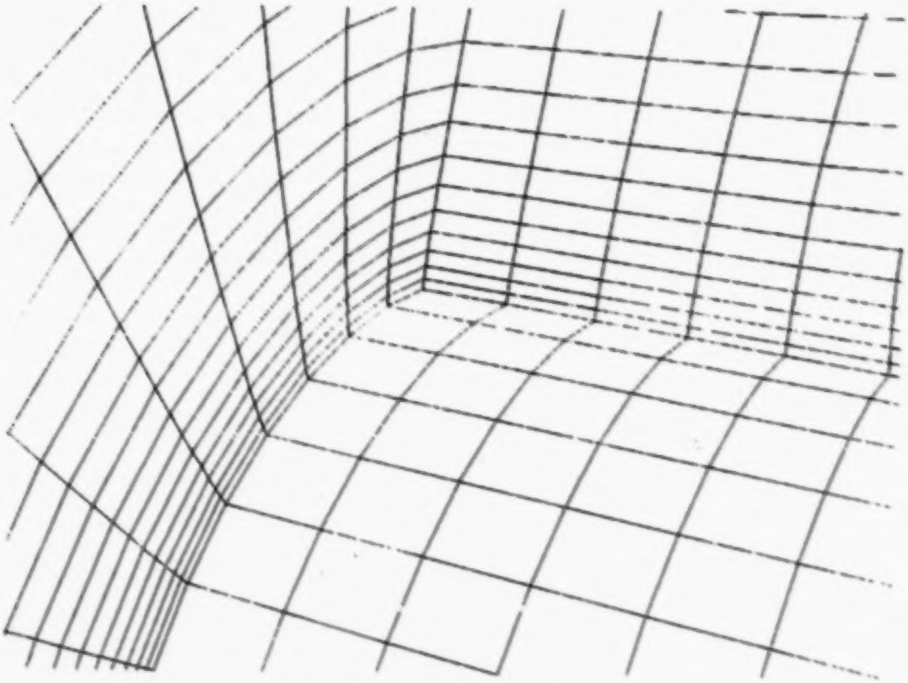
All seven of the treatments for locating boundary points are found in this example—the "read-in-fixed" treatment, the "plane-normal-to" treatment, the "cylinder-about" treatment, the "ellipsoid" treatment, the "collapsed-to-an-axis" treatment, the "collapsed-to-a-point" treatment, and the "match-to-face" treatment.

The convergence history for block 2 of this case was shown in Figure 9. It is difficult to predict a priori how many iterations will be needed to converge a case. The convergence history shows that this case was probably run longer than necessary. Seventy-five or 100 iterations would probably have sufficed for parts (of the iteration schedule) one and two, and part three would have been adequate without the restart. These observations are made by simply looking at the curves and seeing where they flatten out. The restart was accomplished using the file16 input data in Appendix C.

The finished grid is shown in Figures 10c through 10e. It consists of 49,104 points, summed over all three blocks. The grid generation required 103 sec of CPU time on a CRAY X/MP, plus another 24 sec to compile and link 3DGRAPE.



(c) Fuselage, symmetry plane, and selected constant-j surfaces.



(d) Close view of intersection of upper symmetry plane, fuselage in cylindrical region, and constant-j surface.

Figure 10.– Continued

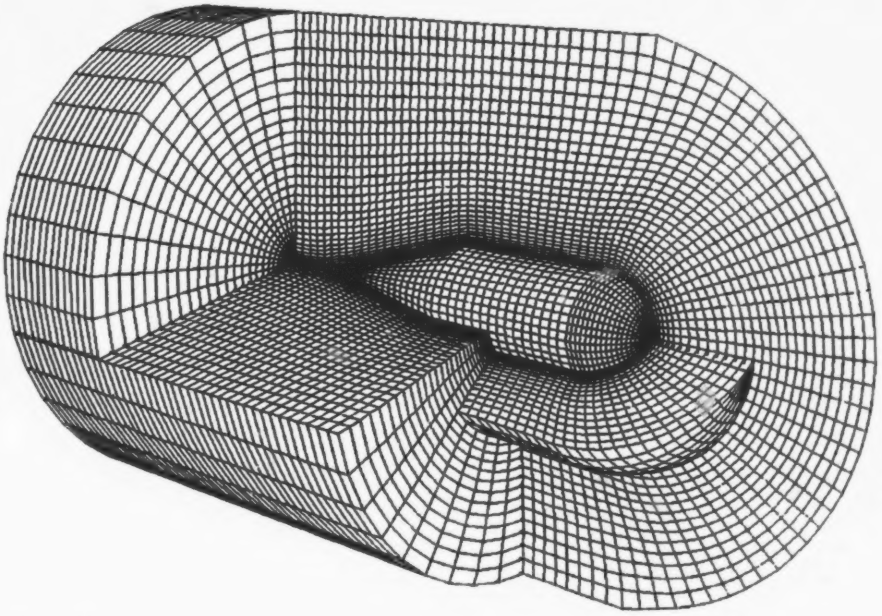


Figure 10e.– Composite view of entire grid.

Figure 10.– Concluded

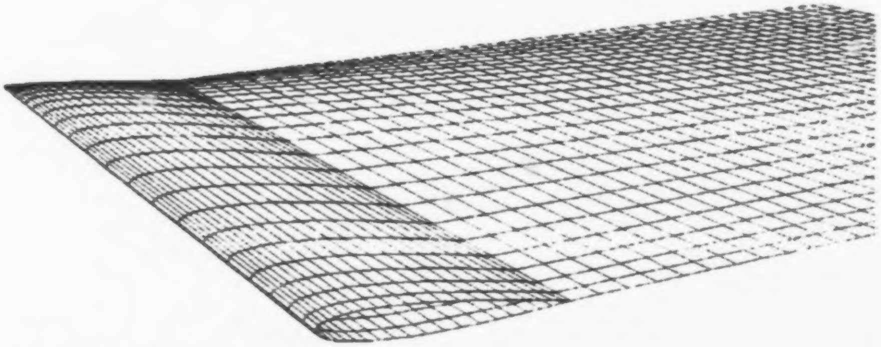
VII. EXAMPLE 2: C-O TYPE GRID ABOUT A WING

Example 2 is a grid about an isolated wing, from a symmetry plane out toward and beyond the tip. The wing has an NACA 0012 airfoil section and a rectangular planform with a 4:1 aspect ratio. The wing is at 5° angle of attack. A flat sheet extends rearward from the wing for five chord lengths. The wing and sheet are shown in figure 11a.

The grid about it is of the C-O type. This nomenclature means that if one were to look at a slice of the grid, taken normal to the span direction, the grid in that slice would be of the C type. If one were to imagine the entire grid reflected about the symmetry plane, giving a grid covering both sides of the wing, and look at a slice of the grid taken normal to the free stream, the grid in that slice would wrap all the way around the wing from tip to tip and back, and thus be of the O type.

Another way to imagine this topology is to first envision a C-type grid about a wing with a sheet behind it (C-type if viewed looking along the span). Assume that grid is chopped off neatly at the tip, producing a C-type grid in the end plane. Then take that end-plane surface, and scribe a line on it running along the outboard edge of the trailing sheet, along the tip, and proceeding forward. Then fold that end plane about the scribed line as though the line were a hinge. Fold both top and bottom halves of the end plane outward until they meet in a horizontal plane. There is no one ideal topology for grids about wings. But this C-O topology promises to be better able to treat blunt leading edges, along with tips which are rounded at the front, than are some other topologies. Its principal virtue here is that it exercises many of the options in 3DGRAPE.

As an elliptic method, 3DGRAPE requires initial conditions. This is especially true of floating boundaries. The main solver in 3DGRAPE which finds points in the interiors of the blocks is extremely robust, but the extrapolation from interior points to boundaries remains problematical. If reasonable initial conditions are not found for those floating boundaries, instability can result, giving a grid which looks like



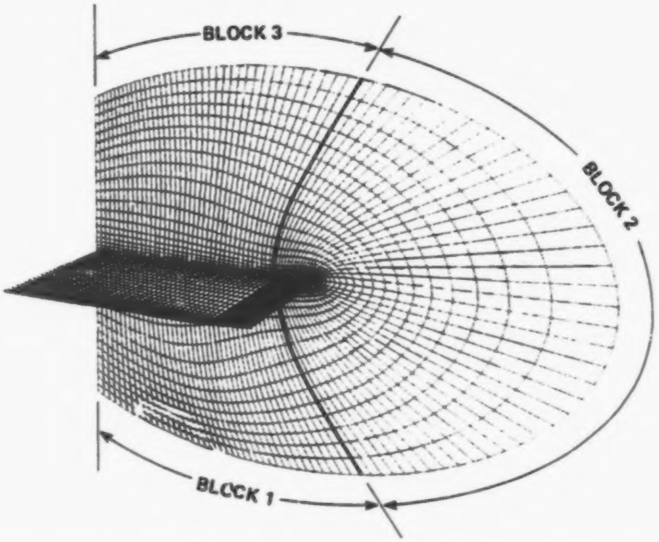
(a) Wing and sheet behind, viewed from root end. NACA 0012 airfoil section, rectangular planform, 4:1 aspect ratio, 5° angle of attack.

Figure 11.– Example 2: C-O type grid about isolated wing.

an explosion in a spaghetti factory. But a dilemma appears in that the better able a program is to make those initial conditions, the more input data it requires. An example of this is the "cylinder-about" input lines. Of the 14 input parameters on those three lines (ignoring keywords), seven are used only in setting the initial conditions. If the writer of such a program is not careful, the supplying of such input data can become a real burden on the user. Therefore, any such program is a tradeoff between burdensome input and instability.

For the reasons set forth in the previous paragraph, this example case was difficult. Finding the control terms and converging to a final grid solution was the easy part; getting a reasonable-looking converged Laplacian solution prior to that was hard. The obvious outer boundary treatment for this topology is an ellipsoid. But as mentioned in a previous chapter, finding initial conditions on ellipsoidal boundaries has proven to be a problem. The method given for locating those outer boundary points involves placing the center of the ellipsoid such that rays can emanate from that center, pass through the "inner boundary" points, pierce the ellipsoid, and thus give the initial locations for the outer boundary points. For outer boundary points connected by grid lines to the wing, this worked as expected. But the inner boundary points on the sheet were unusable in this way; all such rays would have been in a plane and would have pierced the ellipsoid in a line. So another method had to be found for locating outer boundary points connected by grid lines to the sheet.

This alternate method was to break the grid into three blocks—number 1 below the sheet, number 2 wrapping around the wing, and number 3 above the sheet (see fig. 11b). The given method worked for the ellipsoidal outer boundary in block 2. But for blocks 1 and 3, the outer boundary was set to be a cylinder. The proportions of the ellipsoid in block 2 were altered so that it was circular in the y-z (span-vertical) plane, and thus it matched the circular shape of the cylinder at their juncture. A first run was made in this way, using file10 input data shown in Appendix D.



(b) Wing viewed from tip end with symmetry plane, showing block structure.

Figure 11.– Continued

The grid generation was then restarted using file16 input data shown in Appendix F. The desired ellipsoidal outer boundary treatment was specified for all three blocks. At the same time the proportions of the ellipsoid were changed back to the rather eccentric values originally desired, reducing its height from 20 chord lengths to 10. This means that the outer boundary was suddenly brought inside a large number of the interior grid points. This represents a rather extreme use of the restart facility, and shows how powerful it can be. The fact that the solver did not blow up in response to such a shock illustrates that it is robust.

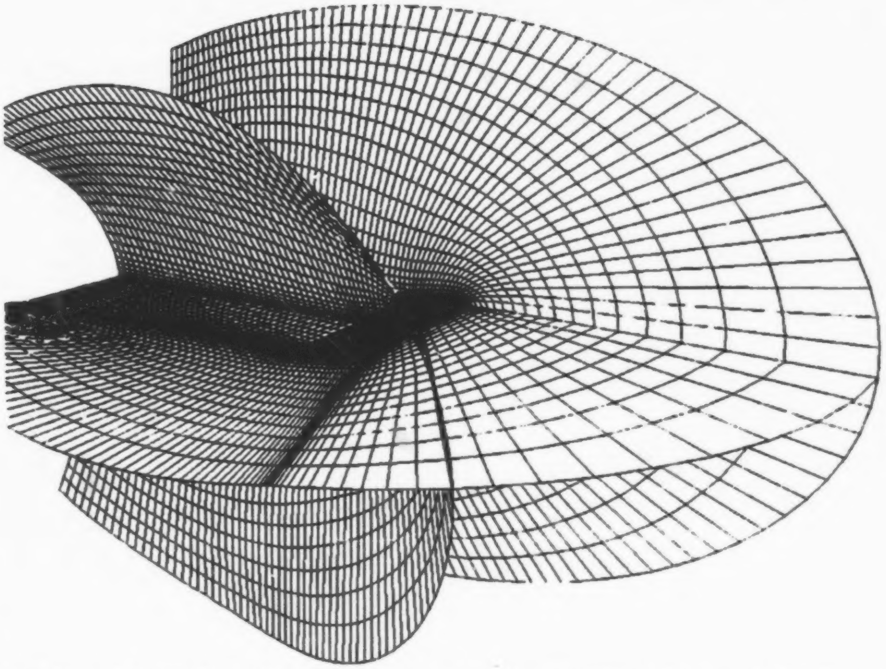
In a fashion similar to the outer boundary treatment, and for the same reasons, the points on the symmetry plane boundary were constrained to lie coincidently on several points during the first run, and then were released to be on a "plane-normal-to" for the restart. The finished symmetry plane grid is seen in figure 11b.

Later in the restart run, a part in the iteration schedule turned control "on," and caused the generation of the RHS terms. The required cell height and near-orthogonality on the wing and sheet resulted (see figs. 11c-e).

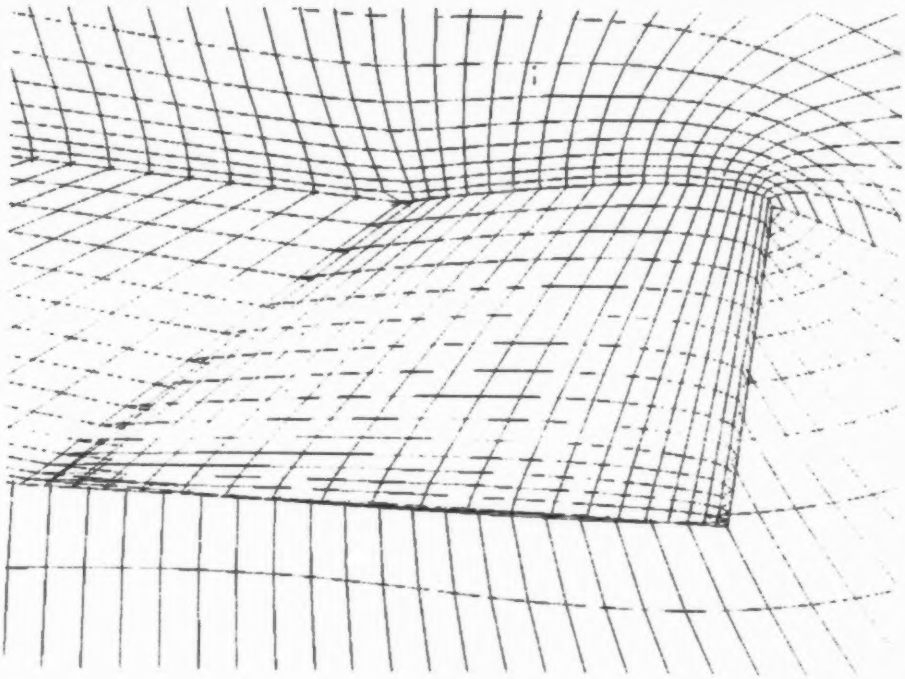
This example includes a rather ambitious use of the "match-to-face" boundary treatment. Face 4 of block 2 is that end plane surface which is folded over on itself. It is divided into two sections, and the two sections match each other. Another option seen in this case, but not in the previous example, is specifying the desired cell height as a piecewise continuous linear function of an index. This is done in face 5 of blocks 1 and 3. A height of 0.04 unit was requested normal to the sheet at the outflow boundary, and 0.02 unit at the trailing edge.

A close examination of this grid will reveal two areas having problems: in the vicinity of the tip at the leading edge, and along the outboard edge of the sheet. For that reason, this grid could not be used exactly as it is in a flow-solver. Those problems are most likely due to the crude body-fitting done by the program in Appendix E. A more sophisticated body-fitting would probably have eliminated the problems, but such a body- fitting program would have been beyond the scope of what could be reproduced in an Appendix. Regardless of those small problem areas, this case does converge, as seen in the plot of the convergence history for block 1 in figures 11f and 11g.

This grid consists of 67,650 points, and required 246 sec of CPU time on a CRAY X/MP. A performance monitor on the X/MP reported that this run, in its entirety (including I/O, setup, etc.), ran at 64.1 MFLOPS.

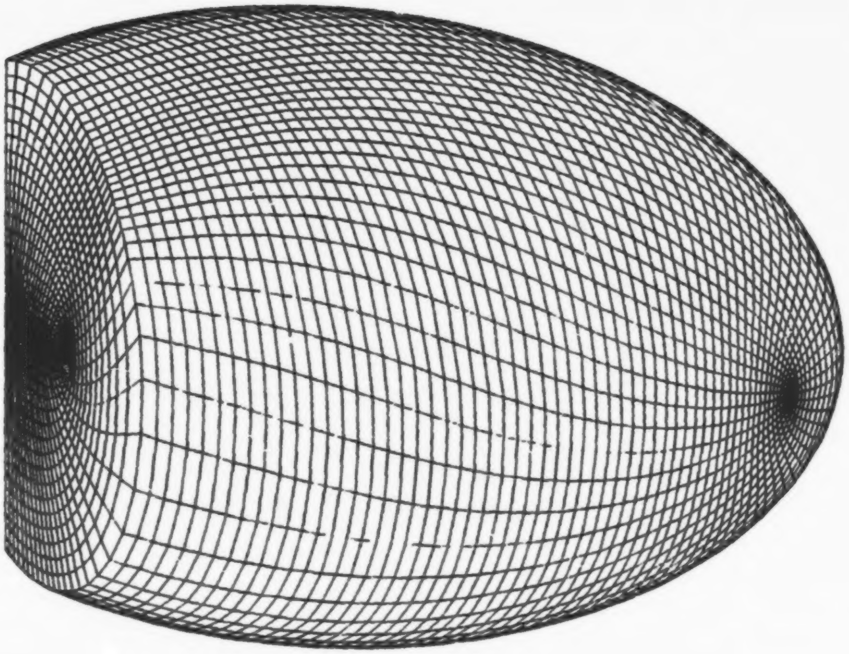


(c) Wing viewed from tip end, with constant-k surface cutting wing, and planform surface.

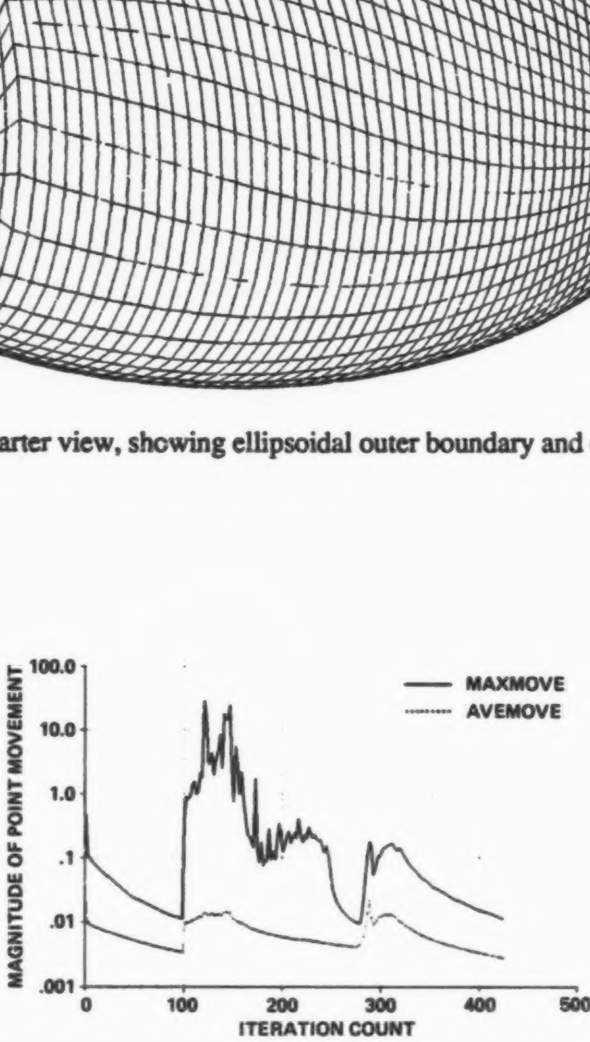


(d) Close-up view of wingtip with constant-k surface cutting wing. Grid cells on wing surface are of constant height and are locally orthogonal.

Figure 11.– Continued

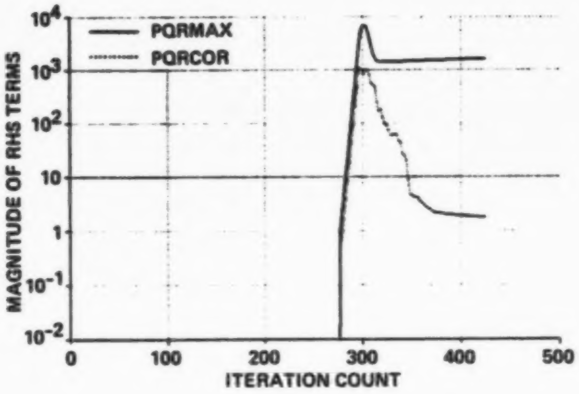


(e) Outboard rear quarter view, showing ellipsoidal outer boundary and outflow boundary plane.



(f) Convergence history for block 1 of example two–Maxmove and Avemove, the magnitude of point movement.

Figure 11.– Continued



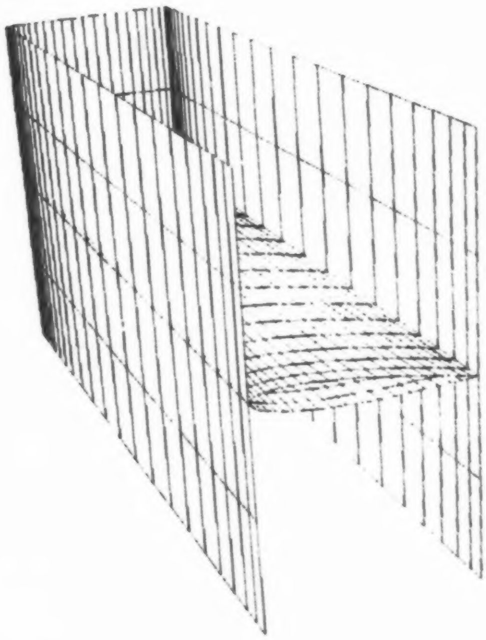
(g) Convergence history for block 1 of example two–Pqrmax and Pqrcor, the magnitude of RHS terms.

Figure 11.– Concluded

VIII. EXAMPLE 3: H-H TYPE GRID ABOUT A WING

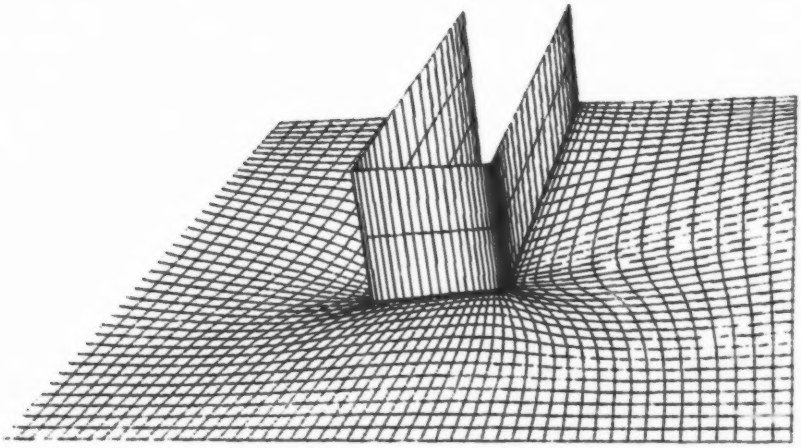
Chapter III discussed a problem which can arise in making a grid of the H-H type about a wing. The problem is the distribution of points in the planform surface. A close examination of figure 4 will reveal that the spacing in the streamwise direction on the wing near the leading edge is fine, as would be expected, but the spacing just upstream of the leading edge is coarse. A discontinuity in spacing should be avoided, but it is especially a problem near the leading edge where flow gradients are steep. Chapter III suggested a solution to this problem which uses 3DGRAPE twice. That approach is illustrated in this example.

The wing is the same one used in the previous example. A small program, listed in Appendix H, writes that wing in a temporary file to be read by a later program. It also identifies the perimeter of the wing, replicates it five times with different vertical biases, producing a vertical wall, and writes that wall in file11 format for a first run of 3DGRAPE. The wing and its wall are shown in figure 12a. The file10 for that run is given in Appendix I. This problem required a topology of five blocks: one directly upstream of the wing, one directly downstream, one directly outboard, one outboard and upstream, and one outboard and downstream. On the three faces consisting of the wall control was activated giving the desired control of spacing around the perimeter of the wing. The third (of five) horizontal surface of that finished grid was extracted for use as the fixed planform surface, and is shown in figure 12b. Note that since the wing is at an angle of attack, the planform surface is not truly a plane.



(a) Wing and walls for use in first run, viewed from root end.

Figure 12.– Example 3: H-H type grid about wing.

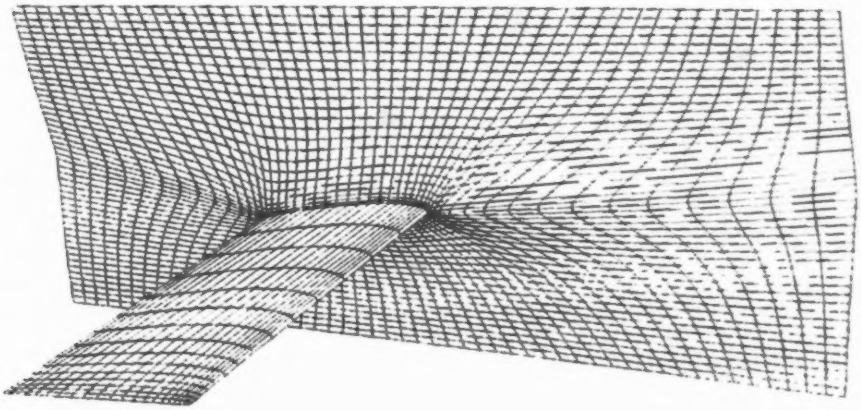


(b) Walls and planform surface resulting from first run, viewed from tip end.

The second small program for this example, given in Appendix J, reads that planform surface, reads the entire wing from the temporary file, combines them into upper and lower planform surfaces, and writes them out in file11 form for the second run of 3DGRAPE. Note that everywhere off the wing the planform surface is double-stored. This grid consists of two blocks—above the planform surface and below it. Since the planform surface is fixed during the second run, there is no communication between the blocks; identical results would have been obtained by generating the upper and lower blocks in two separate runs.

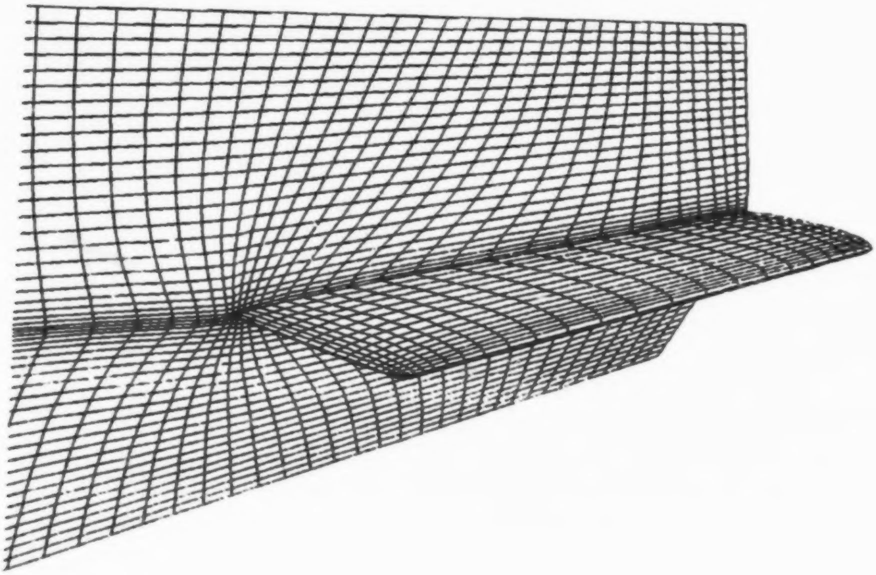
Figures 12c and 12d show the finished grid.

This is the first example case to use the "lightening" feature. Control is lightened along the leading edge and the wingtip in both top and bottom blocks.



(c) Wing and surface normal to span in finished grid.

Figure 12.– Continued



(d) Wing and surface normal to free stream in finished grid.

Figure 12.– Concluded

IX. THEORETICAL DEVELOPMENT

The essence of grid generation is to find a mapping between a certain physical domain and a computational domain. The physical domain is here described by the Cartesian coordinates x,y,z, and the computational domain is described by the uniform orthogonal computational variables ξ,η,ζ. By adjusting that mapping in some appropriate way, it can be arranged that cardinal values of ξ,η,ζ map into the desired values of x,y,z, producing a grid. It is not required that that mapping be given as a simple analytic relationship; a workable situation exists when the mapping is given by equations. If a numerical solution for those equations can be found, the mapping is obtained.

3DGRAPE generates grids by solving a coupled set of Poisson's elliptic partial differential equations in 3-D. Those equations are well known to be

$$\xi_{xx} + \xi_{yy} + \xi_{zz} = P(\xi,\eta,\zeta) \tag{1a}$$

$$\eta_{xx} + \eta_{yy} + \eta_{zz} = Q(\xi,\eta,\zeta) \tag{1b}$$

$$\zeta_{xx} + \zeta_{yy} + \zeta_{zz} = R(\xi,\eta,\zeta) \tag{1c}$$

But those equations in that form are not readily usable. They would require the user to supply boundary conditions for ξ,η,ζ at known values of x,y,z. Users typically want to do the opposite of that; they want to give values for x,y,z at known values of ξ,η,ζ. Equations (1a-c) can be transformed into

$$\alpha_{11}\vec{r}_{\xi\xi} + \alpha_{22}\vec{r}_{\eta\eta} + \alpha_{33}\vec{r}_{\zeta\zeta} + 2(\alpha_{12}\vec{r}_{\xi\eta}\ \alpha_{13}\vec{r}_{\xi\zeta} + \alpha_{23}\vec{r}_{\eta\zeta}) = -J^2(P\vec{r}_\xi + Q\vec{r}_\eta + R\vec{r}_\zeta) \tag{2a}$$

where

$$\vec{r} = \overrightarrow{\begin{bmatrix} x \\ y \\ z \end{bmatrix}} \tag{2b}$$

$$\alpha_{ij} = \sum_{m=1}^{3} \gamma_{mi}\gamma_{mj} \tag{2c}$$

γ_{ij} is the ijth cofactor of the Jacobian matrix M where

$$M = \begin{bmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{bmatrix} \tag{2d}$$

and J is the determinant of M.

A point-SOR solver is used on these equations. The differencing of some first derivatives is adjusted to maximize diagonal dominance and thus enhance stability.

But the real area of interest in elliptic grid generation is the choice of RHS terms. Here they are of the following form:

$$\begin{aligned} P(\xi,\eta,\zeta) = {} & P_1(\eta,\zeta)e^{-a\xi} \\ & + P_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)} \\ & + P_3(\xi,\zeta)e^{-a\eta} \\ & + P_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \\ & + P_5(\xi,\eta)e^{-a\zeta} \\ & + P_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)} \end{aligned} \tag{3}$$

RHS terms Q and R have similar form, using $q_1, q_2, \ldots, q_6$ and $r_1, r_2, \ldots, r_6$. The computational variable ξ has its minimum value, zero, on face 1. It has its maximum value, ξ_{max}, on face 2. ξ is simply j-1. The computational variables η and ζ, along with their maxima η_{max} and ξ_{max} are defined similarly on faces 3-6. It can be seen that the first term in equation (3) is at its maximum on face 1, where ξ=0 and the exponential equals 1, and that that first term decays toward zero with movement into the middle of the block. The free parameter "a" in equation (3) determines the rate of that decay. In a similar fashion the second term is at its maximum on face 2 and decays to zero with movement toward the middle of the block. Similar behavior is seen for the remaining terms in equation (3). The factors $p_1, p_2, \ldots, p_6$ could be thought of as causing the desired clustering and orthogonality near their respective faces, with that influence decaying exponentially with distance from those faces.

The challenge, then, is to find $p_1, p_2, \ldots, p_6, q_1, q_2, \ldots, q_6$ and $r_1, r_2, \ldots, r_6$. If they can be found, the RHS terms over the entire block can be calculated by multiplying by the appropriate exponentials. The method for finding them can best be understood by considering a representative sample, such as p_3, q_3, and r_3, which cause clustering and orthogonality near face 3. At face 3 the third term in equation (3) reduces to just p_3. We also assume that on face 3 all the other terms in equation (3) have vanished, since we are at a distance from their faces and their exponentials have approached zero. We also assume that the Poisson equations (eq. (2)), are valid on face 3. Then if all of the derivatives which make up the left-hand side of equation (2a) can be found, equation (2a) reduces to a 3-x-3 linear system of equations in the unknowns p_3, q_3, and r_3.

The left-hand side of equation (2a) is made up of all possible first and second partial derivatives of $\vec{r}$ with respect to ξ, η, and ζ. Of those derivatives the following are known on face 3 by differencing fixed boundary data: $\vec{r}_\xi$, $\vec{r}_\zeta$, $\vec{r}_{\xi\zeta}$, $\vec{r}_{\xi\xi}$, and $\vec{r}_{\zeta\zeta}$. The derivatives $\vec{r}_\eta$ are found from the desired clustering and orthogonality on face 3. That desired clustering and orthogonality could be specified by the three relations

$$\vec{r}_\xi \cdot \vec{r}_\eta = 0 \tag{4a}$$

$$\vec{r}_\zeta \cdot \vec{r}_\eta = 0 \tag{4b}$$

$$\vec{r}_\eta \cdot \vec{r}_\eta = S^2 \tag{4c}$$

where S is the height to be imposed on the cell on the boundary, illustrated in figure 6. Expansion gives

$$x_\xi x_\eta + y_\xi y_\eta + z_\xi z_\eta = 0 \tag{5a}$$

$$x_\zeta x_\eta + y_\zeta y_\eta + z_\zeta z_\eta = 0 \tag{5b}$$

$$x_\eta^2 + y_\eta^2 + z_\eta^2 = S^2 \tag{5c}$$

Rearranging terms and applying Cramer's rule gives

$$x_\eta = z_\eta(-z_\xi y_\zeta + z_\zeta y_\xi)/D \tag{6a}$$

$$y_\eta = z_\eta(-x_\xi z_\zeta + x_\zeta z_\xi)/D \tag{6b}$$

where

$$D = x_\xi y_\zeta - x_\zeta y_\xi \tag{6c}$$

From equation (2) it can be seen that equations (6a) and (6b) become

$$x_\eta = z_\eta(-\gamma_{12})/(-\gamma_{32}) \tag{7a}$$

$$y_\eta = z_\eta(-\gamma_{22})/(-\gamma_{32}) \tag{7b}$$

Substituting equations (7a) and (7b) into equation (5c) and reducing gives

$$z_\eta = \frac{S\,\gamma_{32}}{\pm\sqrt{\gamma_{12}^2 + \gamma_{22}^2 + \gamma_{32}^2}} \tag{8a}$$

Substituting equation (8a) into equations (7a) and (7b) gives

$$x_\eta = \frac{S\,\gamma_{12}}{\pm\sqrt{\gamma_{12}^2 + \gamma_{22}^2 + \gamma_{32}^2}} \tag{8b}$$

$$y_\eta = \frac{S\,\gamma_{22}}{\pm\sqrt{\gamma_{12}^2 + \gamma_{22}^2 + \gamma_{32}^2}} \tag{8c}$$

Thus we now have values for the derivatives $\vec{r}_\eta$. These can be differenced to obtain $\vec{r}_{\xi\eta}$ and $\vec{r}_{\eta\zeta}$. Note that all derivatives found to here are fixed for all computational time, and thus need to be calculated only once at the start of the run. The only derivatives lacking then in the left-hand side of equation (2a) are $\vec{r}_{\eta\eta}$. These can be found in each iteration by differencing the grid at the present time step. The differencing molecule used for this second derivative is unusual in that it uses an analytic representation for the first derivative, but it can be derived in straightforward manner from the Taylor series. It is

$$\vec{r}_{\eta\eta} = \frac{-7\vec{r}_1 + 8\vec{r}_2 - \vec{r}_3}{2(\Delta\eta)^2} - \frac{3(\vec{r}_\eta)_1}{\Delta\eta} \tag{9}$$

where the numerical subscripts indicate values of the index k, running in the η direction, with k=1 being at face 3. The values for $\vec{r}_\eta$ found in equation (8) are used in equation (9).

We are now ready to solve equation (2a) at face 3 as a linear system. Substituting p_3 for $P(\xi,\eta,\zeta)$, shown above to be a valid substitution on face 3, and similarly q_3 for Q and r_3 for R, equation (2a) becomes

$$P_3\vec{r}_\xi + q_3\vec{r}_\eta + r_3\vec{r}_\zeta = \vec{h} \tag{10a}$$

where

$$\vec{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \vec{f} - \frac{\gamma_{22}}{J^2}\vec{r}_{\eta\eta} \tag{10b}$$

and

$$\vec{f} = (\alpha_{11}\vec{r}_{\xi\xi} + \alpha_{33}\vec{r}_{\zeta\zeta} + 2(\alpha_{12}\vec{r}_{\xi\eta} + \alpha_{13}\vec{r}_{\xi\zeta} + \alpha_{23}\vec{r}_{\eta\zeta}))/(-J^2) \tag{10c}$$

Note that $\vec{f}$ and γ_{22}/J^2 are constant for all computational time, thus the only variable in $\vec{h}$ is $\vec{r}_{\eta\eta}$.

Equation (10a) is just

$$M\begin{bmatrix}p_3\\q_3\\r_3\end{bmatrix} = \vec{h} \tag{11}$$

This system can be solved by Cramer's rule, giving

$$p_3 = (h_1\gamma_{11} + h_2\gamma_{21} + h_3\gamma_{31})/J \tag{12a}$$

$$q_3 = (h_1\gamma_{12} + h_2\gamma_{22} + h_3\gamma_{32})/J \tag{12b}$$

$$r_3 = (h_1\gamma_{13} + h_2\gamma_{23} + h_3\gamma_{33})/J \tag{12c}$$

Thus values for p_3, q_3, and r_3 have been found. RHS terms at the other faces are found similarly. Thus the RHS terms everywhere in the block can be computed. The main iteration loop can be summarized as:

(1) Difference the solution at the current time step to obtain second derivatives at the faces, such as $\vec{r}_{\eta\eta}$ at face 3, using differencing such as in equation (9).

(2) Compute new values for $\vec{h}$ as in equation (10b). From that obtain new values for the RHS terms everywhere in the block.

(3) Take a solution step to update the x,y,z.

(4) Update the locations of points on "floating" boundaries.

Equation (8) includes a choice of sign. That choice is made based on the "handedness" of the block—positive for right-handed and negative for left-handed.

REFERENCES

1. Sorenson, R. L.; and Steger, J. L.: Simplified Clustering of Nonorthogonal Grids Generated by Elliptic Partial Differential Equations, NASA TM-73252, 1977.

2. Thompson, J. F.; Thames, F. C.; and Mastin, C. W.: Automated Numerical Generation of Body-Fitted Curvilinear Coordinate System for Field Containing Any Number of Arbitrary Two-Dimensional Bodies, J. Comp. Phys., vol. 15, no. 3, July 1974, pp. 299-319.

3. Thompson, J. F.; Thames, F. C.; and Mastin, C. W.: TOMCAT–A Code for Numerical Generation of Boundary-Fitted Curvilinear Coordinate Systems on Fields Containing Any Number of Arbitrary Two-Dimensional Bodies, J. Comp. Phys., vol. 24, no. 3, July 1977, pp. 274-302.

4. Steger, J. L.; and Sorenson, R. L.: Automatic Mesh-Point Clustering Near a Boundary in Grid Generation with Elliptic Partial Differential Equations, J. Comp. Phys., vol. 33, no. 3, Dec. 1979, pp. 405-410.

5. Sorenson, R. L.: A Computer Program to Generate Two-Dimensional Grids About Airfoils and Other Shapes by the Use of Poisson's Equation, NASA TM-81198, 1980.

6. Sorenson, R. L.; and Steger, J. L.: Grid Generation in Three Dimensions by Poisson Equations with Control of Cell Size and Skewness at Boundary Surfaces, in Advances in Grid Generation–FED-Vol. 5, K. N. Ghia, ed., ASME, 1983.

7. Sorenson, R. L.: Three-Dimensional Elliptic Grid Generation for an F-16, in Three Dimensional Grid Generation for Complex Configurations–Recent Progress, AGARDograph 309, March 1988, pp. 23-28.

8. Flores, J.; Chaderjian, N. M.; and Sorenson, R. L.: Simulation of Transonic Viscous Flow Over a Fighter-Like Configuration Including Inlet, AIAA Paper 87-1199, June 1987. (Also published in J. Aircraft, vol. 26, no. 4, April 1989.)

9. Sorenson, R. L.: Three-Dimensional Zonal Grids About Arbitrary Shapes by Poisson's Equation, Proc. Second Intern. Conf. Numerical Grid Generation in CFD, Miami, December, 1988 Numerical Grid Generation in Computational Fluid Mechanics, S. Sengupta, J. Hauser, P. R. Eiseman, and C. Taylor, eds., Pineridge Press Ltd., Swansea, U.K., 1988. (Also published as NASA TM-101018, 1988.)

10. Buning, P.; and Steger, J.: Graphics and Flow Visualization in Computational Fluid Dynamics, AIAA-85-1507-CP, July 1985.

11. Walatka, P. P.; and Bunning, P. G.: PLOT3D User's Manual, NASA TM-101067, 1989.

APPENDIX A: FILE10 INPUT DATA FOR EXAMPLE 1

```
run-comment         Example:  hemisphere-cylinder-cone
run-comment         simulation of helicopter fuselage.
number-of-blocks=03-number-of-parts-in-iteration-schedule=03
iterations=020-control=no-coarse/fine=coarse
iterations=150-control=ye-coarse/fine=coarse
iterations=075-control=ye-coarse/fine=fine
filename-11-input=file11ex1      -filename-12-output=
filename-14-grid-output=ex1.bin        -form=3dgrape
write-for-restart=ye-filename-15-output=restartex1
relaxation-param=keep-default


block-01-comment    Hemispherical Nose Cap
dimension-j=019-dimension-k=031-dimension-l=022
handedness=r-initcond=k-cart/sph=spherical
polar-axis=x-along=k-around=l-center=   100.

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
collapsed-to-x-axis-from-x=    0.      -to-x=  -400.     -k-along-
...axis-from-002-to-031-l-around-from-001-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-02-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022

face-3-sections=01-normal=   2.000    -abc=keep-default-light/tight=no
read-in-fixed-xyz-j-from-001-to-019-l-from-001-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
ellipsoid-x-cent=100.        -y-cent=    0.       -z-cent=    0.
...x-semi=  500.       -y-semi=   500.      -z-semi=  500.       -j-from-
...002-to-018-l-from-002-to-021

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031


block-02-comment    Cylinder and cone behind nose
dimension-j=034-dimension-k=031-dimension-l=022
```

```
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-01-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-03-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022

face-3-sections=01-normal=   2.000    -abc=keep-default-light/tight=no
read-in-fixed-xyz-j-from-001-to-034-l-from-001-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
cylinder-about-x-axis-from-x=100.        -to-x=     750.   -j-along-
...axis-from-002-to-033-l-around-from-002-to-021-with-angle=
...-90.         -to-angle= +90.        -radius=     500.

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.         -j-from-002-to-033-k-from-002-
...to-031

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.         -j-from-002-to-033-k-from-002-
...to-031


block-03-comment    Axis downstream
dimension-j=019-dimension-k=031-dimension-l=022
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-02-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=   1125.    -k-from-002-to-031-l-from-001-
...to-022

face-3-sections=02-normal=uncontrolled-abc=keep-default-light/tight=no
collapsed-to-point-x=  750.       -y=    0.        -z=  0.         -with-
...j-from-001-to-001-l-from-001-to-022
collapsed-to-x-axis-from-x=      770.   -to-x=  1125.      -j=along-
...axis-from-002-to-019-l-around-from-001-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
```

```
cylinder-about-x-axis-from-x=  750.       -to-x=     1125.   -j-along-
...axis-from-002-to-018-l-around-from-002-to-021-with-angle=
...-90.         -to-angle= +90.        -radius=     500.

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031
```

APPENDIX B: PROGRAM WHICH MAKES FILE11 DATA FOR EXAMPLE 1

```
c...Program to compute and write body points for the first example case, the
c..."helicopter fuselage," which is actually just a hemisphere-cylinder-cone
c...body.

c...x goes back, y goes out to the side, z goes up.

c...j goes back, k goes out radially, l goes around from bottom to top on the
c...right side.

c...rad is radius of hemisphere, the cylinder, and the cone at its base
c...bl is length of the cylinder
c...cl is length (height?) of the cone

c---------------------------------

      dimension x(85,30), y(85,30), z(85,30)

c----------------------------------

c...Set basic parameters.
      j1=19
      j2a=17
      j2b=18
      lmax=22
      rad=100.
      bl=300.
      cl=350.
      pi=3.141592653589793

c...Define the body, z(x), for negative z, on lower symmetry plane.
      dth=0.5*pi/float(j1-1)
      thet=-dth
      do  1000 j=1,j1
       thet=thet+dth
       x(j,1)=-rad*cos(thet)
       z(j,1)=-rad*sin(thet)
 1000 continue

      dx=bl/float(j2a-1)
      xx=0.
      do  1001 j=j1+1,j1+j2a-1
       xx=xx+dx
       x(j,1)=xx
       z(j,1)=-rad
 1001 continue

       dx=cl/float(j2b-1)
       xx=bl
       do  1002 j=j1+j2a,j1+j2a+j2b-2
        xx=xx+dx
        x(j,1)=xx
        z(j,1)=(xx-cl)*rad/cl-(bl*rad)/cl
 1002 continue

c...Shift it in x so that the origin is at the nose.
       do  1003 j=1,j1+j2a+j2b-2
        x(j,1)=x(j,1)+rad
 1003 continue

c..Rotate z(x) through 180 deg. (+y direction).
       dphi=pi/float(lmax-1)
       phi=0.
       do  1004 l=2, lmax
        phi=phi+dphi
        do  1005 j=1,j1+j2a+j2b-2
         x(j,l)=x(j,1)
         y(j,l)=-z(j,1)*sin(phi)
         z(j,l)= z(j,1)*cos(phi)
 1005  continue
 1004 continue

c...Output in 3DGRAPE's file11 format.
       jl=j1
       j1=j1
       j2end=j1+j2a+j2b-2

       open(unit=11,status='new',form='formatted',file='file11ex1')

       write(11, 2000)
 2000  format('complete-x-for-section-01-of-face-3-of-block-01')
       write(11,100) ((x(j,l),j=1,jl),l=1,lmax)
100    format(6f12.4)
       write(11, 2001)
 2001  format('complete-y-for-section-01-of-face-3-of-block-01')
       write(11,100) ((y(j,l),j=1,jl),l=1,lmax)
       write(11, 2002)
 2002  format('complete-z-for-section-01-of-face-3-of-block-01')
       write(11,100) ((z(j,l),j=1,jl),l=1,lmax)

       write(11, 2003)
 2003 format('complete-x-for-section-01-of-face-3-of-block-02')
      write(11,100) ((x(j,l),j=j1,j2end),l=1,lmax)
      write(11, 2004)
 2004 format('complete-y-for-section-01-of-face-3-of-block-02')
      write(11,100) ((y(j,l),j=j1,j2end),l=1,lmax)
      write(11, 2005)
 2005 format('complete-z-for-section-01-of-face-3-of-block-02')
      write(11,100) ((z(j,l),j=j1,j2end),l=1,lmax)

      close(unit=11)

c...Output in 3DGRAPE's file14 (main grid output) format so that the body alone
c...can be looked-at with grid display graphics.

      open(unit=14,status='new',form='binary',file='lookatit')

      write(14)  2

      write(14)  j1,1,lmax
      write(14)  ((x(j,l),j=1,j1),l=1,lmax),
     1           ((y(j,l),j=1,j1),l=1,lmax),
     2           ((z(j,l),j=1,j1),l=1,lmax)

      write(14)  j2end-j1+1,1,lmax
      write(14)  ((x(j,l),j=j1,j2end),l=1,lmax),
     1           ((y(j,l),j=j1,j2end),l=1,lmax),
     2           ((z(j,l),j=j1,j2end),l=1,lmax)

      close(unit=14)

      stop
      end
```

APPENDIX C: FILE16 INPUT DATA FOR EXAMPLE 1

```
run-comment          Example:  hemisphere-cylinder-cone
run-comment          simulation of helicopter fuselage.
filename-17-input=restartex1
number-of-parts-in-iteration-schedule=01
iterations=025-control=ye-coarse/fine=fine
filename-11-input=file11ex1       -filename-12-output=
filename-14-grid-output=ex1.bin        -form=3dgrape
write-for-restart=no-filename-15-output=restartex1
relaxation-param=keep-default


block-01-comment     Hemispherical Nose Cap

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
collapsed-to-x-axis-from-x=    0.       -to-  -400.        -k-along-
...axis-from-002-to-031-l-around-from-001-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-02-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022

face-3-sections=01-normal=   2.000    -abc=keep-default-light/tight=no
read-in-fixed-xyz-j-from-001-to-019-l-from-001-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
ellipsoid-x-cent=100.        -y-cent=    0.        -z-cent=    0.
...x-semi=  500.      -y-semi=   500.      -z-semi=  500.        -j-from-
...002-to-018-l-from-002-to-021

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031


block-02-comment     Cylinder and cone behind nose

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-01-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022
```

```
face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-03-this-k-from-002-to-031-this-1-from-001-to-022
...-that-k-from-002-to-031-that-1-from-001-to-022

face-3-sections=01-normal=   2.000    -abc=keep-default-light/tight=no
read-in-fixed-xyz-j-from-001-to-034-1-from-001-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
cylinder-about-x-axis-from-x=100.        -to-x=     750.   -j-along-
...axis-from-002-to-033-1-around-from-002-to-021-with-angle=
...-90.        -to-angle= +90.       -radius=     500.

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-033-k-from-002-
...to-031

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-033-k-from-002-
...to-031


block-03-comment    Axis downstream

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-02-this-k-from-002-to-031-this-1-from-001-to-022
...-that-k-from-002-to-031-that-1-from-001-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=   2250.    -k-from-002-to-031-1-from-001-
...to-022

face-3-sections=02-normal=uncontrolled-abc=keep-default-light/tight=no
collapsed-to-point-x=  750.      -y=   0.       -z=  0.          -with-
...j-from-001-to-001-1-from-001-to-022
collapsed-to-x-axis-from-x=     770.   -to-x=  1125.     -j=along-
...axis-from-002-to-019-1-around-from-001-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
cylinder-about-x-axis-from-x=  750.       -to-x=     2250.   -j-along-
...axis-from-002-to-018-1-around-from-002-to-021-with-angle=
...-90.        -to-angle= +90.       -radius=     500.

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-018-k-from-002-
...to-031
```

```
face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.       -j-from-002-to-018-k-from-002-
...to-031
```

APPENDIX D: FILE10 INPUT DATA FOR EXAMPLE 2

```
run-comment          Example:  C-O-Type grid
run-comment          about isolated wing
number-of-blocks=03-number-of-parts-in-iteration-schedule=01
iterations=100-control=no-coarse/fine=coarse
filename-11-input=file11ex2       -filename-12-output=
filename-14-grid-output=ex2.bin         -form=3dgrape
write-for-restart=ye-filename-15-output=restartex2
relaxation-param=keep-default


block-01-comment    Behind and below wing
dimension-j=040-dimension-k=025-dimension-l=022
handedness=r-initcond=l-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  6.        -k-from-001-to-025-l-from-002-
...to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=n
match-to-face-1-block-02-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
collapsed-to-point-x=   3.       -y=  0.        -z= -5.        -with-
...j-from-002-to-039-l-from-002-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-03-this-j-from-002-to-039-this-l-from-002-to-022
...-that-j-from-039-to-002-that-l-from-002-to-022

face-5-sections=01-normal=2-j-stations-abc=keep-default-light/tight=no
norm/sect=001-  .04       -040-  .02
read-in-fixed-xyz-j-from-001-to-040-k-from-001-to-025

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
cylinder-about-x-axis-from-x=    6.       -to-x=       1.   -j-along-
...axis-from-002-to-039-l-around-from-002-to-024-with-angle=
...-90.        -to-angle=   0.       -radius=  10.


block-02-comment    Around the wing
dimension-j=043-dimension-k=025-dimension-l=022
handedness=r-initcond=l-cart/sph=cartesian
```

```
face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-01-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-03-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-3-sections=04-normal=uncontrolled-abc=keep-default-light/tight=no
collapsed-to-point-x=  0.        -y=  0.        -z= -5.        -with-
...j-from-002-to-010-l-from-002-to-022
collapsed-to-point-x= -5.        -y=  0.        -z= -5.        -with-
...j-from-011-to-020-l-from-002-to-022
collapsed-to-point-x= -5.        -y=  0.        -z=  5.        -with-
...j-from-021-to-030-l-from-002-to-022
collapsed-to-point-x=  0.        -y=  0.        -z=  5.        -with-
...j-from-031-to-042-l-from-002-to-022

face-4-sections=02-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-02-this-j-from-002-to-022-this-l-from-002-to-022
...-that-j-from-042-to-022-that-l-from-002-to-022
match-to-face-4-block-02-this-j-from-022-to-042-this-l-from-002-to-022
...-that-j-from-022-to-002-that-l-from-002-to-022

face-5-sections=01-normal=  .02        -abc=keep-default-light/tight=no
read-in-fixed-xyz-j-from-001-to-043-k-from-001-to-025

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
ellipsoid-x-cent=  1.0       -y-cent=   0.       -z-cent=   0.
...x-semi=    8.     -y-semi=    10.     -z-semi=    10.      -j-from-
...002-to-042-k-from-002-to-024


block-03-comment    Behind and above wing
dimension-j=040-dimension-k=025-dimension-l=022
handedness=r-initcond=l-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-02-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  6.        -k-from-001-to-025-l-from-002-
...to-022

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
```

```
collapsed-to-point-x=   3.      -y=  0.        -z=  5.        -with-
...j-from-002-to-039-l-from-002-to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-01-this-j-from-002-to-039-this-l-from-002-to-022
...-that-j-from-039-to-002-that-l-from-002-to-022

face-5-sections=01-normal=2-j-stations-abc=keep-default-light/tight=no
norm/sect=001-  .02       -040-  .04
read-in-fixed-xyz-j-from-001-to-040-k-from-001-to-025

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
cylinder-about-x-axis-from-x=    1.     -to-x=       6.   -j-along-
...axis-from-002-to-039-l-around-from-002-to-024-with-angle=
...  0.        -to-angle=  90.       -radius=  10.
```

APPENDIX E: PROGRAM WHICH MAKES FILE11 DATA FOR EXAMPLE 2

```
c...Program to compute and write body points for the second example case, the
c...C-O-Type grid about an isolated wing.

c...x goes back, y goes out to the side, z goes up.

c...j goes forward along lower surface, around leading-edge, rearward along
c...upper surface.  k is in spanwise direction, l is out from wing to outer
c...boundary.

c----------------------------------------------------------------------------

      dimension x(100),y(100),ramp(100),yequ(100),
     1  xu(100,100),xl(100,100),zu(100,100),zl(100,100)

c----------------------------------------------------------------------------

      bx=1.1
      by=1.03
      jmax=22
      kmax=25
      alpha=5.
      xrot=0.5
      tr=0.12
      span=4.

      pi=3.141592653589793
      jback=40
      xsheet=6.
      yramp=0.94

c------------------- make x-distribution -----------------------------

      bp=bx+1.
      bm=bx-1.
      cl=alog(bp/bm)
      ds=0.4/float(jmax-1)
      s=-ds

      do  1000 j=1,jmax
       s=s+ds
       expa=exp(cl*(4.*s-1.))
       x(j)=(bm-bp*expa)/(-4.*(expa+1.))
 1000 continue
```

C-2

```
c...Now re-scale it to go from 0 to 1+epsilon where epsilon is the
c...increment which will make the NACA 00xx shape close at the t.e.
      factor=(1.+.008930411365)/x(jmax)
      do  1001 j=1,jmax
       x(j)=x(j)*factor
 1001 continue

c-------------  make y-distribution  ------------------------

      bp=by+1.
      bm=by-1.
      cl=alog(bp/bm)
      ds=0.4/float(kmax-1)
      s=0.1-ds

      do  1002 k=1,kmax
       s=s+ds
       expa=exp(cl*(4.*s-1.))
       y(k)=(bm-bp*expa)/(-4.*(expa+1.))
 1002 continue

c...Re-scale it to go from 0 to span.
      factor=span/(y(kmax)-y(1))
      yshift=y(1)
      do  1003 k=1,kmax
       y(k)=(y(k)-yshift)*factor
 1003 continue

c...equi-spaced y for outflow boundary
      dy=span/float(kmax-1)
      yy=-dy
      do  1004 k=1,kmax
       yy=yy+dy
       yequ(k)=yy
 1004 continue

c-----------------------  make ramp for tip  ------------------------

      do  1005 k=1,kmax
       if(y(k).lt.span-yramp) then
        ramp(k)=1.
       else
        ramp(k)=sqrt(1.-((y(k)-span+yramp)**2)/yramp**2)
       endif
 1005 continue
```

```
c-------------------------- make the NACA 00xx profile ----------------

      do  1006 j=1,jmax
       xx=x(j)
       zu(j,1)=5.*tr* (.2969*sqrt(xx) - .126*xx - .3516*xx**2
     1    + .2843*xx**3 - .1015*xx**4)
       zl(j,1)=-zu(j,1)
       do  1007 k=2,kmax
        zu(j,k)=zu(j,1)*ramp(k)
        zl(j,k)=zl(j,1)*ramp(k)
 1007  continue
 1006 continue

c-------------------------- put it at angle of attack --------------------

      alpha=-alpha*pi/180.
      ca=cos(alpha)
      sa=sin(alpha)

      do  1008 k=1,kmax
       do  1009 j=1,jmax
        xx=x(j)
        zz=zu(j,k)
        xu(j,k) = (xx-xrot)*ca + zz*sa + xrot
        zu(j,k) = (xx-xrot)*sa + zz*ca

        zz=zl(j,k)
        xl(j,k) = (xx-xrot)*ca + zz*sa + xrot
        zl(j,k) = (xx-xrot)*sa + zz*ca
 1009  continue
 1008 continue

c--------------------------- write it --------------------------------

      delx=(xsheet-1.)/float(jback-1)
      jmaxmax=2*jmax-1

      open(unit=11,status='new',form='formatted',file='file11ex2')

      write(11, 2000)
 2000 format('complete-x-for-section-01-of-face-5-of-block-01')
      write(11,100)  ((xl(jmax,k)+float(jback-j)*delx,j=1,jback),
     1  k=1,kmax)
100   format(6f12.4)

      write(11, 2001)
 2001 format('complete-y-for-section-01-of-face-5-of-block-01')
      write(11,100)   ((y(k)*float(j-1)/float(jback-1)+
     1   yequ(k)*float(jback-j)/float(jback-1),j=1,jback),k=1,kmax)

      write(11, 2002)
 2002 format('complete-z-for-section-01-of-face-5-of-block-01')
      write(11,100)   ((zl(jmax,k),j=1,jback),k=1,kmax)

c-----------------------------

      write(11, 2003)
 2003 format('complete-x-for-section-01-of-face-5-of-block-02')
      write(11,100)   ((xl(j,k),j=jmax,1,-1),
     1                 (xu(j,k),j=2,jmax),k=1,kmax)

      write(11, 2004)
 2004 format('complete-y-for-section-01-of-face-5-of-block-02')
      write(11,100)   ((y(k),j=1,jmaxmax),k=1,kmax)

      write(11, 2005)
 2005 format('complete-z-for-section-01-of-face-5-of-block-02')
      write(11,100)   ((zl(j,k),j=jmax,1,-1),
     1                 (zu(j,k),j=2,jmax),k=1,kmax)

c----------------------------

      write(11, 2006)
 2006 format('complete-x-for-section-01-of-face-5-of-block-03')
      write(11,100)   ((xu(jmax,k)+float(j-1)*delx,j=1,jback),
     1  k=1,kmax)

      write(11, 2007)
 2007 format('complete-y-for-section-01-of-face-5-of-block-03')
      write(11,100)   ((yequ(k)*float(j-1)/float(jback-1)+
     1   y(k)*float(jback-j)/float(jback-1),j=1,jback),k=1,kmax)

      write(11, 2008)
 2008 format('complete-z-for-section-01-of-face-5-of-block-03')
      write(11,100)   ((zu(jmax,k),j=1,jback),k=1,kmax)

      close(unit=11)

c-----------------------------------------------------------------------------

      open(unit=47,status='new',form='binary',file='lookatit')
      write(47) 1

      write(47) jmaxmax,kmax,1

      write(47) ((xl(j,k),j=jmax,1,-1),
     1            (xu(j,k),j=2,jmax),k=1,kmax),
     2           ((y(k),j=1,jmaxmax),k=1,kmax),
     3           ((zl(j,k),j=jmax,1,-1),
     4            (zu(j,k),j=2,jmax),k=1,kmax)

      close(unit=47)

      stop
      end
```

APPENDIX F: FILE16 INPUT DATA FOR EXAMPLE 2

```
run-comment          Example:  C-O-Type grid
run-comment          about isolated wing.  Restart.
filename-17-input=restartex2
number-of-parts-in-iteration-schedule=02
iterations=175-control=no-coarse/fine=fine
iterations=150-control=ye-coarse/fine=fine
filename-11-input=filellex2      -filename-12-output=
filename-14-grid-output=ex2.bin        -form=3dgrape
write-for-restart=ye-filename-15-output=huh?
relaxation-param=keep-default


block-01-comment    Behind and below wing

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  6.        -k-from-001-to-025-l-from-002-
...to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-02-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.        -j-from-002-to-039-l-from-002-
...to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-03-this-j-from-002-to-039-this-l-from-002-to-022
...-that-j-from-039-to-002-that-l-from-002-to-022

face-5-sections=01-normal=2-j=stations-abc=keep-default-light/tight=no
norm/sect=001-  .04        -040-  .02
read-in-fixed-xyz-j-from-001-to-040-k-from-001-to-025

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
ellipsoid-x-cent=  1.0        -y-cent=   0.        -z-cent=   0.
...x-semi=    8.      -y-semi=    10.      -z-semi=    5.       -j-from-
...002-to-039-k-from-002-to-024


block-02-comment    Around the wing

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-01-this-k-from-001-to-025-this-l-from-002-to-022
```

```
...-that-k-from-001-to-025-that-l-from-002-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-03-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.         -j-from-002-to-042-l-from-002-
...to-022

face-4-sections=02-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-02-this-j-from-002-to-022-this-l-from-002-to-022
...-that-j-from-042-to-022-that-l-from-002-to-022
match-to-face-4-block-02-this-j-from-022-to-042-this-l-from-002-to-022
...-that-j-from-022-to-002-that-l-from-002-to-022

face-5-sections=01-normal=  .02        -abc=keep-default-light/tignt=no
read-in-fixed-xyz-j-from-001-to-043-k-from-001-to-025

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
ellipsoid-x-cent=  1.0        -y-cent=   0.        -z-cent=   0.
...x-semi=    8.      -y-semi=    10.      -z-semi=    5.       -j-from-
...002-to-042-k-from-002-to-024


block-03-comment    Behind and above wing

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-02-this-k-from-001-to-025-this-l-from-002-to-022
...-that-k-from-001-to-025-that-l-from-002-to-022

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  6.         -k-from-001-to-025-l-from-002-
...to-022

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  0.         -j-from-002-to-039-l-from-002-
...to-022

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-01-this-j-from-002-to-039-this-l-from-002-to-022
...-that-j-from-039-to-002-that-l-from-002-to-022

face-5-sections=01-normal=2-j-stations-abc=keep-default-light/tight=no
norm/sect=001-   .02        -040-   .04
read-in-fixed-xyz-j-from-001-to-040-k-from-001-to-025
```

```
face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
ellipsoid-x-cent=  1.0       -y-cent=   0.        -z-cent=   0.
...x-semi=    8.      -y-semi=    10.      -z-semi=    5.       -j-from-
...002-to-039-k-from-002-to-024
```

APPENDIX G: FILE10 INPUT DATA FOR FIRST RUN OF EXAMPLE 3

```
run-comment          H-H type grid about wing
run-comment          First run to make planform surface
number-of-blocks=05-number-of-parts-in-iteration-schedule=02
iterations=150-control=no-coarse/fine=fine
iterations=200-control=ye-coarse/fine=fine
filename-11-input=file11ex3a     -filename-12-output=
filename-14-grid-output=ex3a.bin       -form=3dgrape
write-for-restart=no-filename-15-output=
relaxation-param=keep-default


block-01-comment    In front of wing
dimension-j=016-dimension-k=025-dimension-l=005
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x= -2.        -k-from-001-to-025-l-from-001-
...to-005

face-2-sections=01-normal=2-k-stations-abc=  0.7        -light/tight=no
norm/sect=001-  .020      -025-   .017
read-in-fixed-xyz-k-from-001-to-025-l-from-001-to-005

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  .0        -j-from-002-to-015-l-from-001-
...to-005

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-3-block-04-this-j-from-002-to-015-this-l-from-001-to-005
...-that-j-from-002-to-015-that-l-from-001-to-005

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=  -1.0      -j-from-001-to-015-k-from-002-
...to-024

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=   1.0      -j-from-001-to-015-k-from-002-
...to-024


block-02-comment    Outboard of wing
dimension-j=022-dimension-k=016-dimension-l=005
handedness-r-initcond=k-cart/sph=cartesian
```

```
face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-2-block-04-this-k-from-002-to-016-this-l-from-001-to-005
...-that-k-from-002-to-016-that-l-from-001-to-005

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-1-block-05-this-k-from-002-to-016-this-l-from-001-to-005
...-that-k-from-002-to-016-that-l-from-001-to-005

face-3-sections=01-normal=2-j-stations-abc=  0.7        -light/tight=no
norm/sect=001-  .017      -022-   .029
read-in-fixed-xyz-j-from-001-to-022-l-from-001-to-005

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=    6.0     -j-from-002-to-021-l-from-001-
...to-005

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=  -1.0      -j-from-002-to-021-k-from-002-
...to-015

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=   1.0      -j-from-002-to-021-k-from-002-
...to-015


block-03-comment    Behind wing
dimension-j=016-dimension-k=025-dimension-l=005
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=2-k-stations-abc=  0.7        -light/tight=no
norm/sect=001-  .020      -025-   .029
read-in-fixed-xyz-k-from-001-to-025-l-from-001-to-005

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  3.        -k-from-001-to-025-l-from-001-
...to-005

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  .0        -j-from-002-to-015-l-from-001-
...to-005

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-3-block-05-this-j-from-002-to-015-this-l-from-001-to-005
...-that-j-from-002-to-015-that-l-from-001-to-005

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
```

```
plane-normal-to-z-axis-at-z=  -1.0      -j-from-002-to-016-k-from-002-
...to-024

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=   1.0      -j-from-002-to-016-k-from-002-
...to-024


block-04-comment    In front of and outboard of wing
dimension-j=016-dimension-k=016-dimension-l=005
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x= -2.        -k-from-001-to-016-l-from-001-
...to-005

face-2-sections=02-normal=uncontrolled-abc=keep-default-light/tight=no
read-in-fixed-xyz-k-from-001-to-001-l-from-001-to-005
match-to-face-1-block-02-this-k-from-002-to-016-this-l-from-001-to-005
...-that-k-from-002-to-016-that-l-from-001-to-005

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-01-this-j-from-002-to-015-this-l-from-001-to-005
...-that-j-from-002-to-015-that-l-from-001-to-005

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y= 6.0        -j-from-002-to-015-l-from-001-
...to-005

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=  -1.0      -j-from-002-to-015-k-from-002-
...to-015

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=   1.0      -j-from-002-to-015-k-from-002-
...to-015


block-05-comment    In front of and outboard of wing
dimension-j=016-dimension-k=016-dimension-l=005
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=02-normal=uncontrolled-abc=keep-default-light/tight=no
read-in-fixed-xyz-k-from-001-to-001-l-from-001-to-005
match-to-face-2-block-02-this-k-from-002-to-016-this-l-from-001-to-005
...-that-k-from-002-to-016-that-l-from-001-to-005
```

```
face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  3.        -k-from-001-to-016-l-from-001-
...to-005

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
match-to-face-4-block-03-this-j-from-002-to-015-this-l-from-001-to-005
...-that-j-from-002-to-015-that-l-from-001-to-005

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y= 6.0        -j-from-002-to-015-l-from-001-
...to-005

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=  -1.0      -j-from-002-to-015-k-from-002-
...to-015

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=   1.0      -j-from-002-to-015-k-from-002-
...to-015
```

APPENDIX H: PROGRAM FOR FILE11 FOR FIRST RUN OF EXAMPLE 3

```
c...Program to compute and write just the wing for the third example case,
c...the H-H Type grid about an isolated wing.

c...x goes back, y goes out to the side, z goes up.

c...j goes forward along lower surface, around leading-edge, rearward along
c...upper surface.  k is in spanwise direction, l is out from wing to outer
c...boundary.

c-----------------------------------------------------------------------

      dimension x(100),y(100),ramp(100),
     1  xu(100,100),xl(100,100),zu(100,100),zl(100,100)

c-----------------------------------------------------------------------

      bx=1.1
      by=1.03
      jmax=22
      kmax=25
      alpha=5.
      xrot=0.5
      tr=0.12
      span=4.
      pi=3.141592653589793
      yramp=0.94

c----------------------  make x-distribution  -----------------------

      bp=bx+1.
      bm=bx-1.
      cl=alog(bp/bm)
      ds=0.4/float(jmax-1)
      s=-ds

      do  1000 j=1,jmax
       s=s+ds
       expa=exp(cl*(4.*s-1.))
       x(j)=(bm-bp*expa)/(-4.*(expa+1.))
 1000 continue

c...Now re-scale it to go from 0 to 1+epsilon where epsilon is the
c...increment which will make the NACA 00xx shape close at the t.e.
      factor=(1.+.008930411365)/x(jmax)
      do  1001 j=1,jmax
       x(j)=x(j)*factor
 1001 continue

c---------------------- make y-distribution -----------------------

      bp=by+1.
      bm=by-1.
      c1=alog(bp/bm)
      ds=0.4/float(kmax-1)
      s=0.1-ds

      do  1002 k=1,kmax
       s=s+ds
       expa=exp(c1*(4.*s-1.))
       y(k)=(bm-bp*expa)/(-4.*(expa+1.))
 1002 continue

c...Re-scale it to go from 0 to span.
      factor=span/(y(kmax)-y(1))
      yshift=y(1)
      do  1003 k=1,kmax
       y(k)=(y(k)-yshift)*factor
 1003 continue

c----------------------- make ramp for tip ------------------------

      do  1004 k=1,kmax
       if(y(k).lt.span-yramp) then
        ramp(k)=1.
       else
        ramp(k)=sqrt(1.-((y(k)-span+yramp)**2)/yramp**2)
       endif
 1004 continue

c-------------------------- make the NACA 00xx profile ---------------

      do  1005 j=1,jmax
       xx=x(j)
       zu(j,1)=5.*tr* (.2969*sqrt(xx) - .126*xx - .3516*xx**2
     1    + .2843*xx**3 - .1015*xx**4)
       zl(j,1)=-zu(j,1)
       do  1006 k=2,kmax
        zu(j,k)=zu(j,1)*ramp(k)
        zl(j,k)=zl(j,1)*ramp(k)
 1006  continue
 1005 continue

c-------------------------- put it at angle of attack --------------------

       alpha=-alpha*pi/180.
       ca=cos(alpha)
       sa=sin(alpha)

       do  1007 k=1,kmax
        do  1008 j=1,jmax
         xx=x(j)
         zz=zu(j,k)
         xu(j,k) = (xx-xrot)*ca + zz*sa + xrot
         zu(j,k) = (xx-xrot)*sa + zz*ca

         zz=zl(j,k)
         xl(j,k) = (xx-xrot)*ca + zz*sa + xrot
         zl(j,k) = (xx-xrot)*sa + zz*ca
 1008  continue
 1007 continue

c----------------- write the entire wing to be read later ----------------

       jmaxmax=2*jmax-1

       open(unit=77,status='new',form='binary',file='ex3wing.dat')

       write(77)   jmax,kmax

       write(77)   ((xu(j,k),j=1,jmax),k=1,kmax)
       write(77)   ((y(k),j=1,jmax),k=1,kmax)
       write(77)   ((zu(j,k),j=1,jmax),k=1,kmax)

       write(77)   ((xl(j,k),j=1,jmax),k=1,kmax)
       write(77)   ((y(k),j=1,jmax),k=1,kmax)
       write(77)   ((zl(j,k),j=1,jmax),k=1,kmax)

       close(unit=77)

c---------  write just the perimiter of the wing, stacked lmax times  ------

       lmax=5
       delz=0.5

       open(unit=78,status='new',form='formatted',file='filellex3a')
```

```
      write(78, 2000)
 2000 format('complete-x-for-section-01-of-face-2-of-block-01')
      write(78,100)  ((xu(1,k),k=1,kmax),l=1,lmax)
100   format(6f12.4)

      write(78, 2001)
 2001 format('complete-y-for-section-01-of-face-2-of-block-01')
      write(78,100)  ((y(k),k=1,kmax),l=1,lmax)

      write(78, 2002)
 2002 format('complete-z-for-section-01-of-face-2-of-block-01')
      write(78,100)  ((float(l-3)*delz*abs(float(l-3)/2.)
     1       + (float(l-3)*delz+zu(1,k))*(1.-abs(float(l-3)/2.)),
     2       k=1,kmax),l=1,lmax)

c--------------------------------

      write(78, 2003)
 2003 format('complete-x-for-section-01-of-face-3-of-block-02')
      write(78,100)  ((xu(j,kmax),j=1,jmax),l=1,lmax)

      write(78, 2004)
 2004 format('complete-y-for-section-01-of-face-3-of-block-02')
      write(78,100)  ((y(kmax),j=1,jmax),l=1,lmax)

      write(78, 2005)
 2005 format('complete-z-for-section-01-of-face-3-of-block-02')
      write(78,100)  ((float(l-3)*delz*abs(float(l-3)/2.)
     1       + (float(l-3)*delz+zu(j,kmax))*(1.-abs(float(l-3)/2.)),
     2       j=1,jmax),l=1,lmax)

c--------------------------------

      write(78, 2006)
 2006 format('complete-x-for-section-01-of-face-1-of-block-03')
      write(78,100)  ((xu(jmax,k),k=1,kmax),l=1,lmax)

      write(78, 2007)
 2007 format('complete-y-for-section-01-of-face-1-of-block-03')
      write(78,100)  ((y(k),k=1,kmax),l=1,lmax)

      write(78, 2008)
 2008 format('complete-z-for-section-01-of-face-1-of-block-03')
      write(78,100)  ((float(l-3)*delz*abs(float(l-3)/2.)
     1       + (float(l-3)*delz+zu(jmax,k))*(1.-abs(float(l-3)/2.)),
     2       k=1,kmax),l=1,lmax)
```

```
c---------------------------------

      write(78, 2009)
 2009 format('complete-x-for-section-01-of-face-2-of-block-04')
      write(78,100)   (xu(1,kmax),l=1,lmax)

      write(78, 2010)
 2010 format('complete-y-for-section-01-of-face-2-of-block-04')
      write(78,100)   (y(kmax),l=1,lmax)

      write(78, 2011)
 2011 format('complete-z-for-section-01-of-face-2-of-block-04')
      write(78,100)   (float(l-3)*delz*abs(float(l-3)/2.)
     1        + (float(l-3)*delz+zu(1,kmax))*(1.-abs(float(l-3)/2.)),
     2        l=1,lmax)

c---------------------------------

      write(78, 2012)
 2012 format('complete-x-for-section-01-of-face-1-of-block-05')
      write(78,100)   (xu(jmax,kmax),l=1,lmax)

      write(78, 2013)
 2013 format('complete-y-for-section-01-of-face-1-of-block-05')
      write(78,100)   (y(kmax),l=1,lmax)

      write(78, 2014)
 2014 format('complete-z-for-section-01-of-face-1-of-block-05')
      write(78,100)   (float(l-3)*delz*abs(float(l-3)/2.)
     1        + (float(l-3)*delz+zu(jmax,kmax))*(1.-abs(float(l-3)/2.)),
     2        l=1,lmax)

      close(unit=78)

c----------------------- write it for graphical examination ---------------

      open(unit=79,status='new',form='binary',file='lookatit')

      write(79)  3

      write(79)  jmax,kmax,2

      write(79)  ((xu(j,k),j=1,jmax),k=1,kmax),
     1  ((xl(j,k),j=1,jmax),k=1,kmax),
     2  ((y(k),j=1,jmax),k=1,kmax),
     3  ((y(k),j=1,jmax),k=1,kmax),
     4  ((zu(j,k),j=1,jmax),k=1,kmax),
     5  ((zl(j,k),j=1,jmax),k=1,kmax)

c------------------

      write(79)  2,kmax,lmax

      write(79)  ((xu(1,k),xu(jmax,k),k=1,kmax),l=1,lmax),
     2  ((y(k),y(k),k=1,kmax),l=1,lmax),
     3  ((float(l-3)*delz*abs(float(l-3)/2.)
     4  + (float(l-3)*delz+zu(1,k))*(1.-abs(float(l-3)/2.)),
     5  float(l-3)*delz*abs(float(l-3)/2.)
     7  + (float(l-3)*delz+zu(jmax,k))*(1.-abs(float(l-3)/2.)),
     8  k=1,kmax),l=1,lmax)

c------------------

      write(79) jmax,1,lmax

      write(79) ((xu(j,kmax),j=1,jmax),l=1,lmax),
     1  ((y(kmax),j=1,jmax),l=1,lmax),
     2  ((float(l-3)*delz*abs(float(l-3)/2.)
     3  + (float(l-3)*delz+zu(j,kmax))*(1.-abs(float(l-3)/2.)),
     4  j=1,jmax),l=1,lmax)

      close(unit=79)

      stop
      end
```

APPENDIX I: FILE10 INPUT DATA FOR SECOND RUN OF EXAMPLE 3

```
run-comment         H-H type grid about wing
run-comment         Second run to read planform surface and do it.
number-of-blocks=02-number-of-parts-in-iteration-schedule=03
iterations=030-control=no-coarse/fine=coarse
iterations=170-control=ye-coarse/fine=coarse
iterations=050-control=ye-coarse/fine=fine
filename-11-input=file11ex3b     -filename-12-output=
filename-14-grid-output=ex3b.bin       -form=3dgrape
write-for-restart=no-filename-15-output=
relaxation-param=keep-default


block-01-comment    Above wing
dimension-j=052-dimension-k=040-dimension-l=025
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x= -2.        -k-from-002-to-039-l-from-002-
...to-024

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  3.        -k-from-002-to-039-l-from-002-
...to-024

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  .0        -j-from-001-to-052-l-from-002-
...to-024

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  6.        -j-from-001-to-052-l-from-002-
...to-024

face-5-sections=01-normal=4-j-stations-abc=keep-default-light/tight=ye
norm/sect=001-   .04       -016-   .010     -037-  .010
norm/sect=052-   .04
01-j-lighten-01-k-lighten-no-j-tighten-no-k-tighten
lighten-at-j=016
lighten-at-k=025
read-in-fixed-xyz-j-from-001-to-052-k-from-001-to-040

face-6-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=   1.0      -j-from-001-to-052-k-from-001-
...to-040
```

```
block-02-comment    Below wing
dimension-j=052-dimension-k=040-dimension-l=025
handedness-r-initcond=k-cart/sph=cartesian

face-1-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x= -2.        -k-from-002-to-039-l-from-002-
...to-024

face-2-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-x-axis-at-x=  3.        -k-from-002-to-039-l-from-002-
...to-024

face-3-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  .0        -j-from-001-to-052-l-from-002-
...to-024

face-4-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-y-axis-at-y=  6.        -j-from-001-to-052-l-from-002-
...to-024

face-5-sections=01-normal=uncontrolled-abc=keep-default-light/tight=no
plane-normal-to-z-axis-at-z=  -1.0      -j-from-001-to-052-k-from-001-
...to-040

face-6-sections=01-normal=4-j-stations-abc=keep-default-light/tight=ye
norm/sect=001-  .04       -016-   .010     -037-  .010
norm/sect=052-  .04
01-j-lighten-01-k-lighten-no-j-tighten-no-k-tighten
lighten-at-j=016
lighten-at-k=025
read-in-fixed-xyz-j-from-001-to-052-k-from-001-to-040
```

APPENDIX J: PROGRAM FOR FILE11 FOR SECOND RUN OF EXAMPLE 3

```
c...Program to exerpt the planform surface from the first run for example 3,
c...read the wing surface, combine them, and output file11 input for the
c...second run of example 3.  The lmaxs had all better be the same.

      dimension xp(100,50),yp(100,50),zp(100,50)

      dimension x1(40,40),y1(40,40),z1(40,40),
     1  x2(40,40),y2(40,40),z2(40,40), x3(40,40),y3(40,40),z3(40,40),
     2  x4(40,40),y4(40,40),z4(40,40), x5(40,40),y5(40,40),z5(40,40)

      dimension xu(25,30),yu(25,30),zu(25,30),
     1  xl(25,30),yl(25,30),zl(25,30)

c---------------------------------------------------------------------------

      open(unit=50,status='old',form='binary',file='ex3a.bin')

      read(50)   maxblk

      read(50)   jmax1,kmax1,lmax1

      lplan=(lmax1+1)/2

      read(50)   (((dummy,j=1,jmax1),k=1,kmax1),l=1,lplan-1),
     1            ((x1(j,k),j=1,jmax1),k=1,kmax1),
     2           (((dummy,j=1,jmax1),k=1,kmax1),l=lplan+1,lmax1),
     3           (((dummy,j=1,jmax1),k=1,kmax1),l=1,lplan-1),
     4            ((y1(j,k),j=1,jmax1),k=1,kmax1),
     5           (((dummy,j=1,jmax1),k=1,kmax1),l=lplan+1,lmax1),
     6           (((dummy,j=1,jmax1),k=1,kmax1),l=1,lplan-1),
     7            ((z1(j,k),j=1,jmax1),k=1,kmax1),
     8           (((dummy,j=1,jmax1),k=1,kmax1),l=lplan+1,lmax1)

      read(50)   jmax2,kmax2,lmax2

      read(50)   (((dummy,j=1,jmax2),k=1,kmax2),l=1,lplan-1),
     1            ((x2(j,k),j=1,jmax2),k=1,kmax2),
     2           (((dummy,j=1,jmax2),k=1,kmax2),l=lplan+1,lmax1),
     3           (((dummy,j=1,jmax2),k=1,kmax2),l=1,lplan-1),
     4            ((y2(j,k),j=1,jmax2),k=1,kmax2),
     5           (((dummy,j=1,jmax2),k=1,kmax2),l=lplan+1,lmax1),
     6           (((dummy,j=1,jmax2),k=1,kmax2),l=1,lplan-1),
     7            ((z2(j,k),j=1,jmax2),k=1,kmax2),
     8           (((dummy,j=1,jmax2),k=1,kmax2),l=lplan+1,lmax1)
```

```
      read(50)  jmax3,kmax3,lmax3

      read(50)  (((dummy,j=1,jmax3),k=1,kmax3),l=1,lplan-1),
     1           ((x3(j,k),j=1,jmax3),k=1,kmax3),
     2          (((dummy,j=1,jmax3),k=1,kmax3),l=lplan+1,lmax1),
     3          (((dummy,j=1,jmax3),k=1,kmax3),l=1,lplan-1),
     4           ((y3(j,k),j=1,jmax3),k=1,kmax3),
     5          (((dummy,j=1,jmax3),k=1,kmax3),l=lplan+1,lmax1),
     6          (((dummy,j=1,jmax3),k=1,kmax3),l=1,lplan-1),
     7           ((z3(j,k),j=1,jmax3),k=1,kmax3),
     8          (((dummy,j=1,jmax3),k=1,kmax3),l=lplan+1,lmax1)

      read(50)  jmax4,kmax4,lmax4

      read(50)  (((dummy,j=1,jmax4),k=1,kmax4),l=1,lplan-1),
     1           ((x4(j,k),j=1,jmax4),k=1,kmax4),
     2          (((dummy,j=1,jmax4),k=1,kmax4),l=lplan+1,lmax1),
     3          (((dummy,j=1,jmax4),k=1,kmax4),l=1,lplan-1),
     4           ((y4(j,k),j=1,jmax4),k=1,kmax4),
     5          (((dummy,j=1,jmax4),k=1,kmax4),l=lplan+1,lmax1),
     6          (((dummy,j=1,jmax4),k=1,kmax4),l=1,lplan-1),
     7           ((z4(j,k),j=1,jmax4),k=1,kmax4),
     8          (((dummy,j=1,jmax4),k=1,kmax4),l=lplan+1,lmax1)

      read(50)  jmax5,kmax5,lmax5

      read(50)  (((dummy,j=1,jmax5),k=1,kmax5),l=1,lplan-1),
     1           ((x5(j,k),j=1,jmax5),k=1,kmax5),
     2          (((dummy,j=1,jmax5),k=1,kmax5),l=lplan+1,lmax1),
     3          (((dummy,j=1,jmax5),k=1,kmax5),l=1,lplan-1),
     4           ((y5(j,k),j=1,jmax5),k=1,kmax5),
     5          (((dummy,j=1,jmax5),k=1,kmax5),l=lplan+1,lmax1),
     6          (((dummy,j=1,jmax5),k=1,kmax5),l=1,lplan-1),
     7           ((z5(j,k),j=1,jmax5),k=1,kmax5),
     8          (((dummy,j=1,jmax5),k=1,kmax5),l=lplan+1,lmax1)

      close(unit=50)

c--------------------------------

      open(unit=51,status='old',form='binary',file='ex3wing.dat')

      read(51) jmaxw,kmaxw

      read(51) ((xu(j,k),j=1,jmaxw),k=1,kmaxw),
     1          ((yu(j,k),j=1,jmaxw),k=1,kmaxw),
     2          ((zu(j,k),j=1,jmaxw),k=1,kmaxw)

      read(51) ((xl(j,k),j=1,jmaxw),k=1,kmaxw),
     1          ((yl(j,k),j=1,jmaxw),k=1,kmaxw),
     2          ((zl(j,k),j=1,jmaxw),k=1,kmaxw)

      close(unit=51)

c-----------------------------------

      open(unit=52,status='new',form='formatted',file='filellex3b')

      write(52, 2000)
 2000 format('complete-x-for-section-01-of-face-5-of-block-01')
      write(52,100)  ((xl(j,k),j=1,jmaxl), (xu(j,k),j=2,jmaxw),
     1  (x3(j,k),j=2,jmax3), k=1,kmax3),
     2  ((x4(j,k),j=1,jmax4), (x2(j,k),j=2,jmax2),
     3  (x5(j,k),j=2,jmax5), k=2,kmax2)
100   format(6f12.4)

      write(52, 2001)
 2001 format('complete-y-for-section-01-of-face-5-of-block-01')
      write(52,100)  ((yl(j,k),j=1,jmaxl), (yu(j,k),j=2,jmaxw),
     1  (y3(j,k),j=2,jmax3), k=1,kmax3),
     2  ((y4(j,k),j=1,jmax4), (y2(j,k),j=2,jmax2),
     3  (y5(j,k),j=2,jmax5), k=2,kmax2)

      write(52, 2002)
 2002 format('complete-z-for-section-01-of-face-5-of-block-01')
      write(52,100)  ((zl(j,k),j=1,jmaxl), (zu(j,k),j=2,jmaxw),
     1  (z3(j,k),j=2,jmax3), k=1,kmax3),
     2  ((z4(j,k),j=1,jmax4), (z2(j,k),j=2,jmax2),
     3  (z5(j,k),j=2,jmax5), k=2,kmax2)

c-----------------------------------

      write(52, 2003)
 2003 format('complete-x-for-section-01-of-face-6-of-block-02')
      write(52,100)  ((xl(j,k),j=1,jmaxl), (xl(j,k),j=2,jmaxw),
     1  (x3(j,k),j=2,jmax3), k=1,kmax3),
     2  ((x4(j,k),j=1,jmax4), (x2(j,k),j=2,jmax2),
     3  (x5(j,k),j=2,jmax5), k=2,kmax2)

      write(52, 2004)
 2004 format('complete-y-for-section-01-of-face-6-of-block-02')
```

```
      write(52,100)  ((y1(j,k),j=1,jmax1), (y1(j,k),j=2,jmaxw),
     1  (y3(j,k),j=2,jmax3), k=1,kmax3),
     2  ((y4(j,k),j=1,jmax4), (y2(j,k),j=2,jmax2),
     3  (y5(j,k),j=2,jmax5), k=2,kmax2)

      write(52, 2005)
 2005 format('complete-z-for-section-01-of-face-6-of-block-02')
      write(52,100)  ((z1(j,k),j=1,jmax1), (z1(j,k),j=2,jmaxw),
     1  (z3(j,k),j=2,jmax3), k=1,kmax3),
     2  ((z4(j,k),j=1,jmax4), (z2(j,k),j=2,jmax2),
     3  (z5(j,k),j=2,jmax5), k=2,kmax2)

      close(unit=52)

      write(*, 2006) jmax1+jmaxw+jmax3-2, kmax3+kmax2-1
 2006 format(/'Planform grid has jmax=',i3,3x,'kmax=',i3)

      stop
      end
```

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-102224	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle

The 3DGRAPE Book: Theory, Users' Manual, Examples

5. Report Date

July 1989

6. Performing Organization Code

7. Author(s)

Reese L. Sorenson

8. Performing Organization Report No.

A-89176

10. Work Unit No.

505-60

9. Performing Organization Name and Address

Ames Research Center
Moffett Field, CA 94035

11. Contract or Grant No.

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546-0001

13. Type of Report and Period Covered

Technical Memorandum

14. Sponsoring Agency Code

15. Supplementary Notes

Point of Contact: Reese L. Sorenson, Ames Research Center, MS 258-1
Moffett Field, CA 94035 (415) 694-4471 or FTS 464-4471

16. Abstract

This document is a users' manual for a new three-dimensional grid generator called 3DGRAPE. The program, written in FORTRAN, is capable of making zonal (blocked) computational grids in or about almost any shape. Grids are generated by the solution of Poisson's differential equations in three dimensions. The program automatically finds its own values for inhomogeneous terms which give near-orthogonality and controlled grid cell height at boundaries. Grids generated by 3DGRAPE have been applied to both viscous and inviscid aerodynamic problems, and to problems in other fluid-dynamic areas. The smoothness for which elliptic methods are known is seen here, including smoothness across zonal boundaries.

An introduction giving the history, motivation, capabilities, and philosophy of 3DGRAPE is presented first. Then follows a chapter on the program itself. The input is then described in detail. A chapter on reading the output and debugging follows. Three examples are then described, including sample input data and plots of output. Last is a chapter on the theoretical development of the method.

17. Key Words (Suggested by Author(s)) Grid generation Mesh generation Computational fluid dynamics CFD	18. Distribution Statement Unclassified-Unlimited Subject Category – 01

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	119	A06

NASA FORM 1626 OCT 86 For sale by the National Technical Information Service, Springfield, Virginia 22161